МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет	*Кафедра*
вычислительной техники	МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

Направленность Программная инженерия

БАКАЛАВРСКАЯ РАБОТА

на тему

Веб-приложение для администрирования корпоративного мессенджера

Студент _____ Новосельцев Александр Андреевич
(подпись, дата) (ФИО полностью)

Руководитель _____ Балашова И.Ю.
(подпись, дата) (фамилия, инициалы)

Нормоконтролер _____ Такташкин Д.В.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____№_____)

Заведующий кафедрой _____ Козлов А.Ю.
(подпись) (фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____№____)

Секретарь ГЭК _____ Попова Н.А.
(подпись) (фамилия, инициалы)

Пенза, 2026

Федеральное государственное бюджетное образовательное учреждение высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет	*Кафедра*
Вычислительной техники	МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____А.Ю. Козлов
«26» декабря 2025 г

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА

1. Студент____Новосельцев Александр Андреевич___гр.__22ВП1__факультета ВТ
<Фамилия, имя, отчество полностью>

2.Тема работы_Веб-приложение для администрирования корпоративного мессенджера

Тема утверждена приказом ПГУ №_____от "_____"_____20_____

2. Руководитель работы___к.т.н., доцент кафедры «МО и ПЭВМ» Балашова И.Ю._____

3. Задание на работу (назначение разработки, исходные данные и т.п.)_____

1) Назначение: панель администратора корпоративного мессенджера

2) Среда выполнения: Windows 11.

Языковые средства разработки: TypeScript.

Инструментальные средства: Zustand, React, TanStack Query, Tailwind CSS, Visual Paradigm 17.1, Visual Studio Code.

Технологии разработки: функциональное программирование, компонентный подход

3) Основные функции приложения:

а) авторизация;

б) просмотр информации о пользователях, рабочих пространствах и жалобах;

в) добавление, изменение, удаление пользователей, рабочих пространств;

д) мониторинг состояния системы.

4. Перечень подлежащих разработке вопросов

1) *Анализ предметной области и постановка задачи;*
2) *Проектирование;*
3) *Реализация;*
4) *Контроль качества программного обеспечения;*
5) *Оформление пояснительной записки.*

5. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	07.04.2026 – 13.04.2026	
2	Проектирование программных средств	20%	14.04.2026 – 25.04.2026	
3	Реализация программных средств	30%	27.04.2026 – 11.05.2026	
4	Тестирование программных средств	20%	12.05.2026 – 21.05.2026	
5	Оформление пояснительной записки	20%	22.05.2026 – 02.06.2026	

Дата выдачи задания «26» декабря 2025г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан ФВТ _____ Л.Р. Фионова
(подпись, дата)

Реферат

Пояснительная записка содержит 109 страниц, 22 рисунка, 7 таблиц, 20 использованных источников, 5 листингов и 5 приложений.

АДМИНИСТРИРОВАНИЕ, ПАНЕЛЬ АДМИНИСТРАТОРА, ВЕБ-ПРИЛОЖЕНИЕ, GRAFANA, TANSTACK QUERY, TYPESCRIPT, UML, REACT, SPA, ZUSTAND, ВИЗУАЛИЗАЦИЯ ДАННЫХ

Объект исследования – веб-приложение для администрирования корпоративного мессенджера.

Цель выпускной квалификационной работы – спроектировать и разработать веб-приложение для администрирования корпоративного мессенджера.

Технологии разработки – среда разработки – Visual Studio Code, язык программирования – TypeScript с библиотекой React, для работы с данными использовались Zustand и TanStack Query.

Среда исполнения – веб-браузер, Windows 11.

Результат работы – в ходе выполнения выпускной квалификационной работы разработано веб-приложение для администрирования корпоративного мессенджера.

Содержание

Введение 7

1 Анализ предметной области и постановка задачи на разработку веб-приложения для администрирования корпоративного мессенджера 9

1.1 Анализ предметной области 9

1.2 Анализ средств администрирования мессенджеров 10

1.2.1 Mattermost 10

1.2.2 VK Teams 11

1.2.3 Rocket.Chat 12

1.2.4 Сравнительный анализ инструментов 13

1.3 Анализ требований 17

1.3.1 Функциональные требования 17

1.3.2 Нефункциональные требования 20

1.4 Планирование разработки и оценка бюджета 22

2 Проектирование приложения для администрирования корпоративного мессенджера 25

2.1 Проектирование архитектуры панели администратора 25

2.2 Проектирование пользовательских сценариев 25

2.3 Концептуальная модель приложения 27

2.4 Проектирование пользовательского интерфейса 30

3 Реализация панели администратора корпоративного мессенджера 33

3.1 Реализация общей структуры и архитектуры приложения 33

3.2 Реализация пользовательского интерфейса 35

3.2.1 Общий макет и стилизация 35

3.2.2 Раздел управления рабочими пространствами 36

3.2.3 Раздел обработки жалоб пользователей 38

3.2.4 Вспомогательные экраны 39
3.3 Организация обмена данными с серверной частью 40
3.4 Аутентификация и управление состоянием сеанса 41
3.4.1 Хранение сеанса администратора 41
3.4.2 Защита маршрутов 42
3.4.3 Серверное и локальное состояние экранов 43
4 Тестирование приложения администрирования корпоративного мессенджера 45
4.1 Тестирование методом «белого ящика» 45
4.1.1 Покрытие операторов 47
4.1.2 Покрытие решений 48
4.1.3 Комбинаторное покрытие условий 48
4.2 Модульное тестирование 50
4.3 Оценка качества кода 52
4.3.1 SLOC тестирование 52
4.3.2 Метрики Холстеда 53
4.3.3 Метрика Чепина 55
4.3.4 Автоматизированная проверка 56
Заключение 59
Список использованных источников 61
Приложение А (справочное) Глоссарий 63
Приложение Б (рекомендуемое) Графическая часть 66
Приложение В (обязательное) Листинг клиентского приложения 75
Приложение Г (рекомендуемое) Листинг инструментов тестирования 97
Приложение Д (рекомендуемое) Акт о внедрении 107

Введение

Развитие корпоративных коммуникаций в последние годы все чаще опирается на специализированные мессенджеры, обеспечивающие быстрый обмен сообщениями, организацию рабочих пространств, совместную работу в чатах и управление задачами. Однако надежная работа такого программного комплекса невозможна без удобного и безопасного администрирования: контроля пользователей и их ролей, конфигурации рабочих пространств мониторинга обращений и инцидентов, а также поддержки эксплуатационных настроек системы. В условиях роста числа сотрудников и увеличения объема данных ручное администрирование становится трудоемкими и приводит к повышению риска ошибок, поэтому актуальной является разработка веб-панели администрирования как отдельного, целенаправленного инструмента управления корпоративным мессенджером.

Объектом исследования выступает процесс администрирования корпоративной системы обмена сообщениями и связанных модулей (рабочие пространства, пользователи, жалобы, конфигурация).

Предметом исследования являются методы средства проектирования и реализации веб-приложения административной панели, обеспечивающего удобный интерфейс управления и корректное взаимодействие с серверной частью по API.

Цель работы – спроектировать и реализовать веб-приложение панели администратора корпоративного мессенджера, обеспечивающее выполнение ключевых административных функций через единый интерфейс с соблюдением требований безопасности, удобства и расширяемости.

Для достижения поставленной цели необходимо решить следующие задачи:

– проанализировать назначение административных функций в корпоративном мессенджере и сформировать требования к панели администратора;

– спроектировать структуру и навигацию веб-приложения, определить основные сценарии работы администратора;

– определить модель взаимодействия клиентской части с сервером, форматы данных и обработку ошибок;

– реализовать клиентское веб-приложение панели администратора;

– обеспечить корректную интеграцию с API корпоративного мессенджера и единые подходы к запросам;

– провести проверку работоспособности.

Практическая значимость работы заключается в создании готового к использованию веб-инструмента, который упрощает сопровождение корпоративного мессенджера: сокращает время на рутинные операции, снижает вероятность ошибок, обеспечивает централизованный контроль и делает эксплуатацию системы более прозрачной.

В результате выполнения работы будет получено веб-приложение панели администратора, предоставляющее администратору набор ключевых функций управления корпоративным мессенджером и интегрированное с серверными сервисами через документированные API.

1 Анализ предметной области и постановка задачи на разработку веб-приложения для администрирования корпоративного мессенджера

1.1 Анализ предметной области

Предметная область работы – администрирование корпоративного мессенджера: совокупность процессов, данных и ролей, связанных с эксплуатацией системы корпоративных коммуникаций и совместной работы. В широком смысле корпоративный мессенджер объединяет обмен сообщениями, организацию рабочих пространств, участие пользователей в чатах, управление задачами и каналами обратной связи. В узком смысле для данной работы предметная область ограничивается тем, что доступно и управляемо администратором через отдельное веб-приложение панели администратора, взаимодействующее с серверной частью по REST API.

Актуальность темы обусловлена устойчивым ростом спроса организаций на собственные каналы цифровой коммуникации [1]. Переход к удаленной и гибридной работе, увеличение числа распределенных команд и объема обмена сообщениями делают корпоративные мессенджеры неотъемлемой частью ИТ-инфраструктуры предприятия наряду с почтой и системами совместной работы. При этом масштабирование таких систем сопровождается усложнением администрирования: возрастает число пользователей, рабочих пространств, ролей и политик доступа, а ошибки конфигурации напрямую влияют на безопасность данных и непрерывность бизнес-процессов.

Граница «клиент мессенджера для сотрудника» и «панель администратора» важна: обычный пользователь решает задачи общения и работы в рабочем пространстве; администратор – задачи создания и сопровождения инфраструктуры мессенджера, контроля конфигурации и обработки административных обращений.

Ниже приведены основные понятия и термины, необходимые для лучшего понимания предметной области и более успешной дальнейшей работы над приложением.

Чат – это средство общения в сети Интернет, позволяющее двум и более участникам обмениваться сообщениями в режиме реального времени [2].

Рабочее пространство – изолированный контур, в котором находятся чаты, задачи и участники, относящиеся к конкретной организации.

Тариф – система ставок платы за различные услуги [3].

Жалоба – обращение пользователя с описанием проблемы или нарушения; для администратора – поток входящих сообщений.

Роль в рабочем пространстве – определенный набор прав и полномочий, которые позволяют разграничивать доступ к функционалу участникам и настраивать права взаимодействия в зависимости от их статуса или задач.

В результате анализа предметной области были выделены основные объекты исследуемой темы, а также понятия и термины, необходимые для проектирования, а также дальнейшей разработки и сопровождения приложения. Кроме того, была обоснована актуальность выделения отдельного веб-интерфейса для эксплуатации мессенджера в контуре организации.

1.2 Анализ средств администрирования мессенджеров

1.2.1 Mattermost

Mattermost представляет собой корпоративную платформу для командной коммуникации, ориентированную на организации, которым требуется высокий уровень контроля над инфраструктурой, данными и политиками безопасности. В отличие от полностью облачных мессенджеров, Mattermost активно используется в формате self-hosted-решения, то есть может быть развернут на серверах самой организации. Это делает систему особенно подходящей для компаний с повышенными требованиями к защите информации, внутренним регламентам и независимости от внешних поставщиков [4].

Для настройки параметров системы и выполнения иных административных задач здесь предусмотрен веб-интерфейс System Console. Интерфейс System Console представлен на рисунке 1.

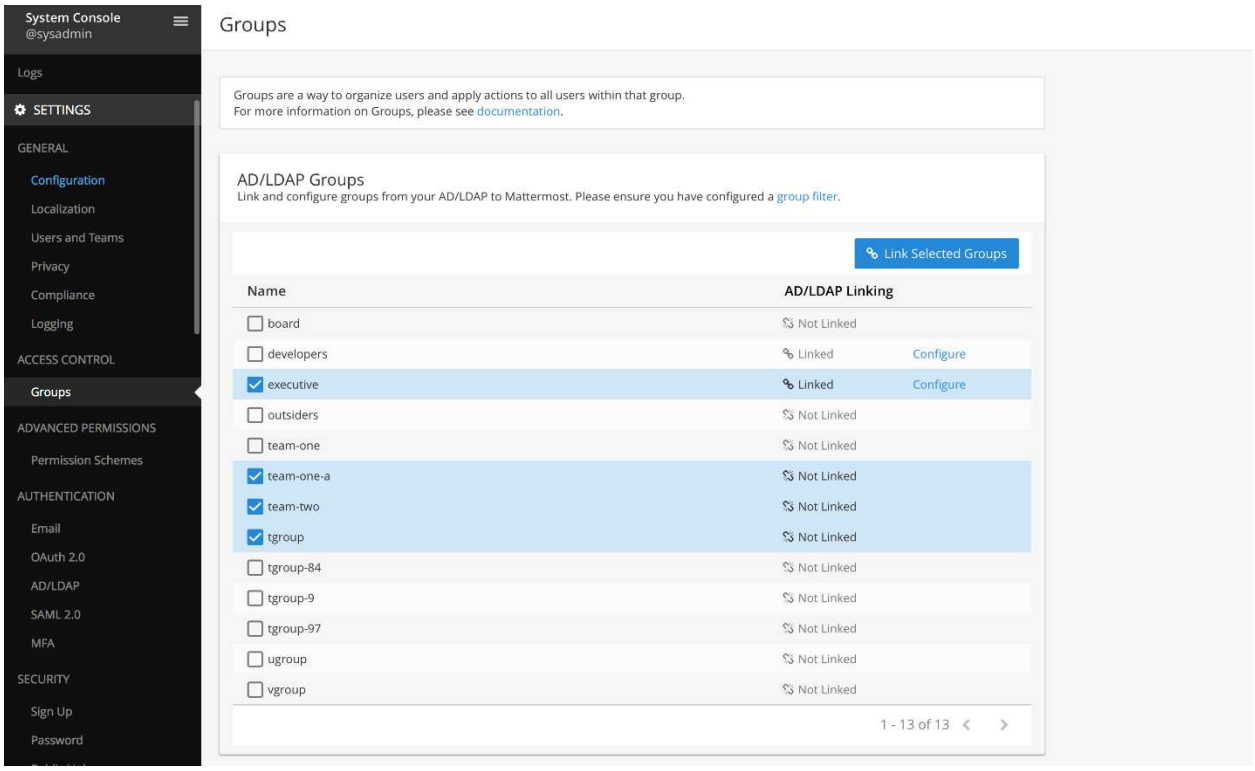

Рисунок 1 – Интерфейс System Console

Помимо этого, при делегировании прав пользователям нужно проявлять осторожность, так как при выдаче некоторых прав, пользователь может получить доступ к функционалу, выходящему за рамки роли.

1.2.2 VK Teams

VK Teams – корпоративный мессенджер и система видеоконференцсвязи в составе платформы VK WorkSpace [5]. Администрирование мессенджера выполняется не изолированным приложением, а через единую панель администратора VK WorkSpace, которая объединяет настройки почты, мессенджера, учетный записей и других сервисов экосистемы.

Такой подход характерен для крупных корпоративных внедрений: администратор работает в одном веб-интерфейсе, переключаясь между разделами. Для организаций с требованиями к локальному размещению данных предусмотрена on-premise-поставка с полным циклом установки и интеграции панели с сервисами мессенджера.

В основные разделы панели входят:

– управление аккаунтами;

– управление группами и каналами;

– управление мини-приложениями.

Интерфейс панели администратора представлен на рисунке 2.

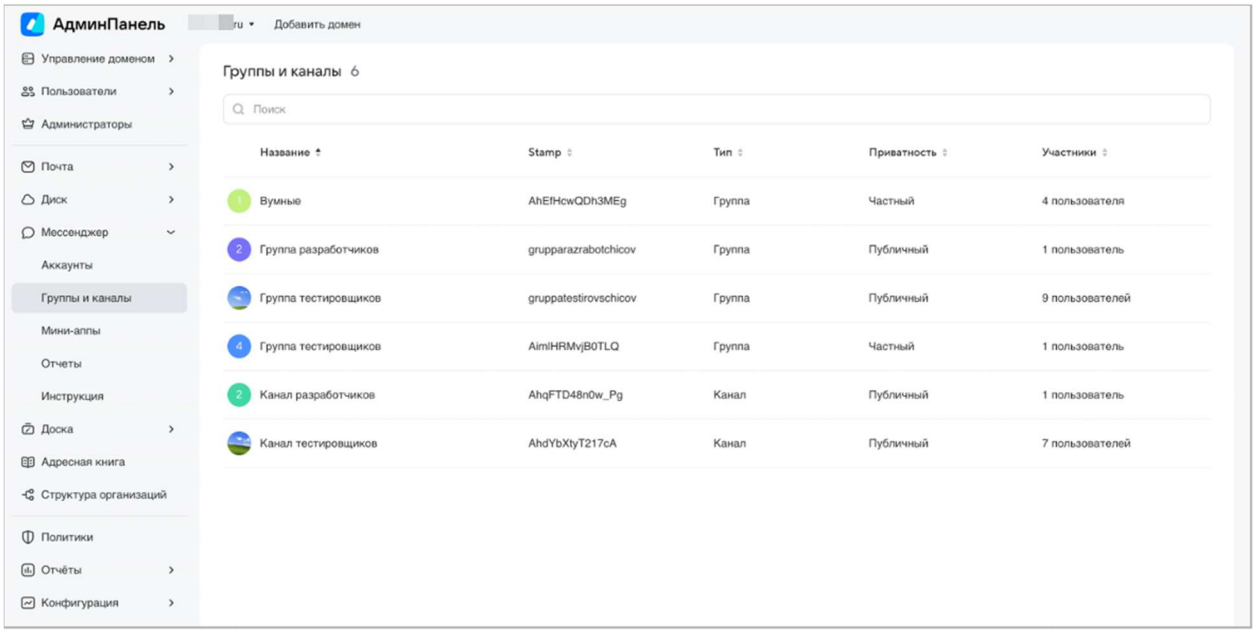

Рисунок 2 – Интерфейс панели администратора VK Teams

К положительным сторонам панели администратора можно отнести такие моменты как: развитая ролевая модель, наличие инструментов управления сессиями, группами, каналами и политиками автодобавления, а также возможность расширения функционала с помощью мини-приложений.

Среди отрицательных сторон можно выделить закрытость экосистемы, избыточность для небольших внедрений и высокий порог внедрения.

1.2.3 Rocket.Chat

Rocket.Chat – платформа корпоративных коммуникаций с открытым исходным кодом. Администрирование строится вокруг понятия рабочего пространства: один развернутый сервер может обслуживать организацию через рабочее пространство с пользователями, ролями, каналами и настройками [6].

Администратор рабочего пространства получает доступ к разделам Manage – Workspace в веб-интерфейсе, где предоставлены возможности для

широкого спектра задач: от первичной настройки до мониторинга и расширения функциональности.Интерфейс Rocket.Chat представлен на рисунке 3.

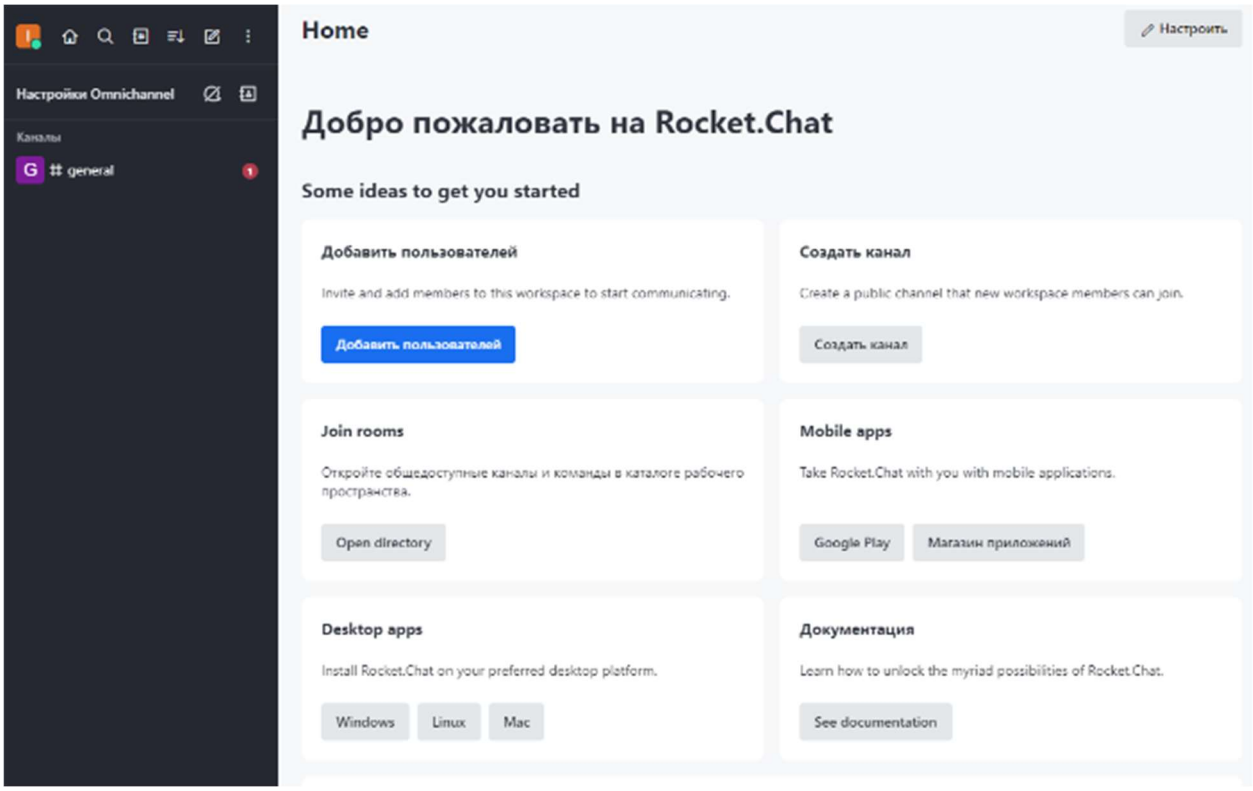

Рисунок 3 – Средства администрирования Rocket.Chat

В сильный сторонах можно отметить развитую экосистему приложений и интеграций, понятную модель рабочего пространства как единицы администрирования, а также открытый код.

В слабых же сторонах отмечается сложность администрирования при масштабировании и кластеризации, слабые инструменты аналитики.

1.2.4 Сравнительный анализ инструментов

Для полного анализа средств администрирования нужно провести сравнение рассмотренных инструментов по определенным критериям.Наиболее важные для работы аспекты описаны ниже.

К1 – управление организационной структурой. Здесь оценивается, насколько полно средство администрирования позволяет создавать настраивать и сопровождать организационные контуры работы – рабочие пространства,

команды, группы, политики членства и связанные с ними ограничения. Так же при оценке учитываются групповые политики, возможности создания и удаления контуров, назначения ответственных (руководитель, владелец, администратор) и привязки тарифов, лицензий или квот к контуру.

К2 – управление пользователями и ролями. Оценивается возможность администрировать учетные записи, их жизненный цикл и ролевую модель – как на уровне всей системы, так и внутри отдельных контуров.

К3 – управление коммуникациями. Оценивается администрирование каналов коммуникации – чатов, каналов, групп и правил доступа к ним.

К4 – обратная связь и модерация. Оценивается наличие встроенных средств для работы с обращениями пользователей – жалобы, претензии, инциденты, модерация контента – и поддержки их обработки.

К5 – мониторинг и состояние системы. Оцениваются возможности администратора по контролю работоспособности платформы и изменению эксплуатационных настроек приложения. Здесь рассматриваются инструменты проверки статуса сервисов, доступности API, просмотра логов и системных параметров, диагностики ошибок.

К6 – аналитика и отчетность. Оценивается наличие аналитических данных об использовании мессенджера и возможностей их агрегации и выгрузки. При оценке учитываются инструменты ведения статистики пользователей, сообщений, каналов, активности, а также возможность экспорта данных.

К7 – безопасность. Оценивается совокупность средств защиты доступа, аудита и соответствия корпоративным и отраслевым требованиям.

К8 – удобство интерфейса панели администратора. Оценивается эргономика и обучаемость административного веб-интерфейса – насколько быстро администратор выполняет типовые задачи без ошибок и обращения к документации.

К9 – расширяемость и интеграция. Оценивается возможность расширять функциональность мессенджера и интегрировать его с внешними системами без изменения ядра продукта.

К10 – автономное развертывание. Оценивается возможность развернуть и эксплуатировать систему в инфраструктуре организации без обязательной привязки к облаку вендора.

Оценка по критериям будет проводиться по шкале от 0 до 2, где 0 – возможность отсутствует либо не выявлена в открытой документации и интерфейсе администрирования, 1 – возможность реализована частично: доступна с ограничениями, требует внешних инструментов, зависит от тарифа/поставки или покрывает сценарий не полностью, 2 – возможность реализована как штатная административная функция и явно описана в документации или доступна в интерфейсе.

В таблице 1 приведено сравнение рассмотренных инструментов по описанным выше критериям.

Таблица 1 – Оценка инструментов по критериям

Критерий	VK Teams	Rocket.Chat	Mattermost
К1	2 – есть централизованное управление пользователями организации.	2 – имеется административный раздел пользователей, импорт, блокировка и настройка учетных записей.	2 – есть управление пользователями, командами и приглашениями через системную консоль.
К2	1 – роли есть, но гибкость настройки зависит от корпоративной конфигурации.	2 – поддерживается развитая система ролей и разрешений.	2 – есть системные и командные роли, права настраиваются через консоль администратора.
К3	1 – организация и группы управляются централизованно, но модель рабочего пространства выражена не как отдельный универсальный объект.	2 – команды, каналы и пользователи администрируются через веб-интерфейс.	2 – команды и каналы являются базовыми сущностями и управляются через системную консоль.
К4	1 – отдельный workflow жалоб выражен ограниченно, чаще через поддержку организации.	1 – есть модерация и администрирование сообщений, но полноценный модуль жалоб требует настройки.	1 – есть модерация и управление контентом, но отдельный сценарий жалоб не является центральной функцией.

Продолжение таблицы 1

Критерий	VK Teams	Rocket.Chat	Mattermost
К5	1 – в облачной версии детали инфраструктуры скрыты, доступны в основном административные сведения.	2 – при self-hosted доступны интеграции наблюдаемости и мониторинг.	2 – для self-hosted доступны системная консоль, журналы и интеграции мониторинга.
К6	1 – административные журналы доступны, но детализация зависит от тарифа и поставки.	2 – журналы и аудит действий доступны в enterprise/self-hosted сценариях.	2 – есть журналы, аудит и системные сведения, особенно в enterprise/self-hosted конфигурации.
К7	2 – интерфейс ориентирован на корпоративного администратора и встроен в платформу.	1 – web-интерфейс подробный, но может быть перегружен настройками.	2 – системная консоль структурирована по разделам и удобна для администрирования.
К8	1 – интеграции есть, но в основном в рамках корпоративной экосистемы.	2 – высокая расширяемость: webhooks, приложения, marketplace, API.	2 – поддерживаются плагины, webhooks, API и интеграции с корпоративными сервисами.
К9	2 – есть корпоративные механизмы безопасности и управление доступом.	2 – есть SSO, 2FA, ограничения доступа и политики безопасности.	2 – поддерживаются SSO, MFA, политики доступа и настройки безопасности сервера.
К10	1 – документация есть, но часть сведений привязана к корпоративной платформе и тарифам.	2 – документация по администрированию и self-hosted развертыванию подробная.	2 – подробная документация системной консоли, развертывания и администрирования.

По результатам сравнения наиболее широкие возможности администрирования показывают Mattermost и Rocket.Chat. Это связано с тем, что оба решения ориентированы не только на использование как готового мессенджера, но и на самостоятельное развёртывание, интеграцию и настройку корпоративной инфраструктуры. Mattermost получает немного более высокую оценку за развитую системную консоль и документацию по администрированию. Rocket.Chat близок к нему по функциональности, особенно в части интеграций и гибкой настройки прав.

VK Teams уступает по ряду критериев не из-за отсутствия корпоративных функций, а из-за иной модели поставки: значительная часть администрирования зависит от облачной платформы, тарифа и внутренних механизмов экосистемы. Так, на данной платформе возможности самостоятельного мониторинга, расширения и детального администрирования выражены менее явно.

1.3 Анализ требований

1.3.1 Функциональные требования

По результатам анализа предметной области и изучения инструментов администрирования в некоторых популярных мессенджерах был получен список необходимых функциональных требований к разрабатываемому приложению.

Функциональные требования описывают, какие действия должна выполнять система и какие возможности она должна предоставлять пользователям. Они фиксируют ожидаемое поведение программного продукта с точки зрения его функций: какие данные система принимает, как их обрабатывает, какие результаты возвращает и какие сценарии работы поддерживает.

Функциональные требования формируются на основе целей проекта, анализа предметной области и сценариев взаимодействия пользователей с системой. В них отражаются роли пользователей, доступные им операции, ограничения доступа и ожидаемые результаты выполнения действий.

Такие требования являются основой для проектирования архитектуры, базы данных, пользовательского интерфейса и API. На их основе определяются основные модули системы, связи между сущностями, состав экранов приложения и перечень программных методов, которые должны быть реализованы. Кроме того, функциональные требования используются при тестировании: каждое требование можно сопоставить с одним или несколькими сценариями проверки, чтобы убедиться, что система действительно выполняет заявленные функции.

Для визуализации полученных требований была использована диаграмма вариантов использования в нотации UML [7]. Диаграмма вариантов использования для разрабатываемого приложения представлена на рисунке 4.

Рисунок 4 – Диаграмма вариантов использования

Для более детального рассмотрения системы ниже описаны спецификации некоторых вариантов использования.

На рисунке 5 представлена спецификация прецедента входа в систему.

Прецедент: вход в систему
ID: 1
Краткое описание: администратор проходит аутентификацию в панели администратора
Актеры: администратор
Предусловие: 1. администратор не авторизован в системе; 2. сервер аутентификации доступен.
Основной поток: 1. администратор вводит логин и пароль; 2. администратор нажимает на кнопку входа.
Постусловие: 1. администратор авторизован в системе; 2. администратор перенаправлен на главную страницу панели.

Рисунок 5 – Спецификация прецедента входа в систему

На рисунке 6 представлена спецификация прецедента создания рабочего пространства.

Прецедент: создать рабочее пространство
ID: 2
Краткое описание: администратор регистрирует новое рабочее пространство в системе.
Актеры: администратор
Предусловие: 1. администратор авторизован в системе; 2. администратор находится на странице создания рабочего пространства; 3. в системе существует хотя бы один тарифный план и пользователь, которого можно назначить руководителем.
Основной поток: 1. администратор указывает название, руководителя, тариф; 2. нажимает на кнопку создания.
Постусловие: 1. новое рабочее пространство создано; 2. администратор перенаправлен на страницу списка рабочих пространств.

Рисунок 6 – Спецификация прецедента создания рабочего пространства

На рисунке 7 представлена спецификация прецедента просмотра жалобы.

Прецедент: просмотреть жалобу
ID: 3
Краткое описание: администратор просматривает жалобу
Актеры: администратор
Предусловие: 1. администратор авторизован в системе; 2. администратор находится на странице жалоб; 3. в системе существует хотя бы одна жалоба.
Основной поток: 1. администратор просматривает список; 2. нажимает на кнопку создания.
Альтернативный поток: 1.1. выбирает значение в фильтре для отображения жалоб с определенным статусом
Постусловие: 1. выведена основная информация по жалобе.

Рисунок 7 – Спецификация прецедента просмотра жалобы

Таким образом, в результате анализа функциональных требований была составлена диаграмма вариантов использования, описывающая основные способы взаимодействия клиента с приложением. Полученные данные требуются для проведения анализа нефункциональных требований и помогут при проектировании системы.

1.3.2 Нефункциональные требования

На основе полученных ранее данных можно провести анализ нефункциональных требований. Нефункциональные требования определяют качественные характеристики веб-приложения панели администратора: не состав функций, а условия, при которых эти функции должны выполняться

надежно, безопасно и удобно для администратора. Ниже приведены основные требования, предъявляемые приложению.

Панель администратора предназначена для управления корпоративным мессенджером и обрабатывает чувствительные данные, поэтому доступ к приложению должен быть строго ограничен: доступ ко всем разделам должен быть доступен только успешной аутентификации администратора, пароли и токены не должны сохраняться в открытом виде. Окончательная проверка прав остается на стороне сервера – клиент должен лишь отражать результат и не подменять серверную авторизацию.

Интерфейс должен быть понятен и удобен для использования даже администратору без глубоких технических знаний. Для этого все основные разделы должны быть доступны из единой навигации за не более чем два перехода. Терминология, использованная в приложении, должна быть согласована с предметной областью.

Помимо простоты использования в приложении должны быть предусмотрены возможности для легкого и быстрого расширения функционала. Для этого при разработке приложений будет использоваться наиболее известный инструмент для разработки веб-приложений React. Использование данной библиотеки позволит существенно снизить трудоемкость расширения функционала [8].

Несмотря на эпизодическое использование панели, от нее ожидается быстрая реакция при работе с данными. Для этого при взаимодействии с большими списками применяется постраничная выдача, чтобы не перегружать систему и сервер. На время выполнения запросов отображаются индикаторы загрузки, а повторное нажатие кнопок отправки запроса блокируется. Помимо этого, допускается кратковременное кэширование данных на клиенте, что позволяет снизить число обращений к API без ущерба для актуальности после изменений.

Функционал панели администратора напрямую зависит от доступности API-шлюза и микросервисов корпоративного мессенджера. При недоступности

сервиса или сетевой ошибке пользователю должно отображаться понятное сообщение, а возможность повторить запрос должно быть доступна без перезагрузки всего приложения. Операция считается выполненной только после подтверждения сервером, успех до этого момента приложением не должен отображаться. Введенные в форму данные не теряются до отправки. После успешного создания или изменения сущности списки и детальные представления обновляются, чтобы отображать состояние, согласованное с сервером. Таким образом, панель корректно ведет себя в условиях частичных сбоев распределенной системы, характерных для микросервисной архитектуры.

Так, в результате анализа нефункциональных требований были рассмотрены основные критерии и характеристики, на основе которых и будет вестись разработка панели администратора. Помимо этого, полученные данные потребуются и при проектировании приложения.

1.4 Планирование разработки и оценка бюджета

Управление сроками и затратами – неотъемлемая часть создания программного продукта. Для разрабатываемого веб-приложения заранее нужно зафиксировать: какие работы выполняются, в какой последовательности, кто их делает и сколько времени на них уходит. Без такого плана сложно согласовать разработку клиентской части и уложиться в установленный срок.

Для построения календарного графика и первичной оценки затрат был использован Microsoft Project. Это программное обеспечение для управления, предназначенное для планирования задач, распределения ресурсов и затрат на них, отслеживания прогресса выполнения работ, а также аналитики рабочих нагрузок, бюджетов и построения визуальных графиков, а также отчетов [9]. Программа подходит для индивидуальной и корпоративной работы благодаря поддержке различных методологий управления и возможности формирования отчетов.

С помощью Project для проекта была составлена диаграмма Ганта, представленная на рисунке 8.

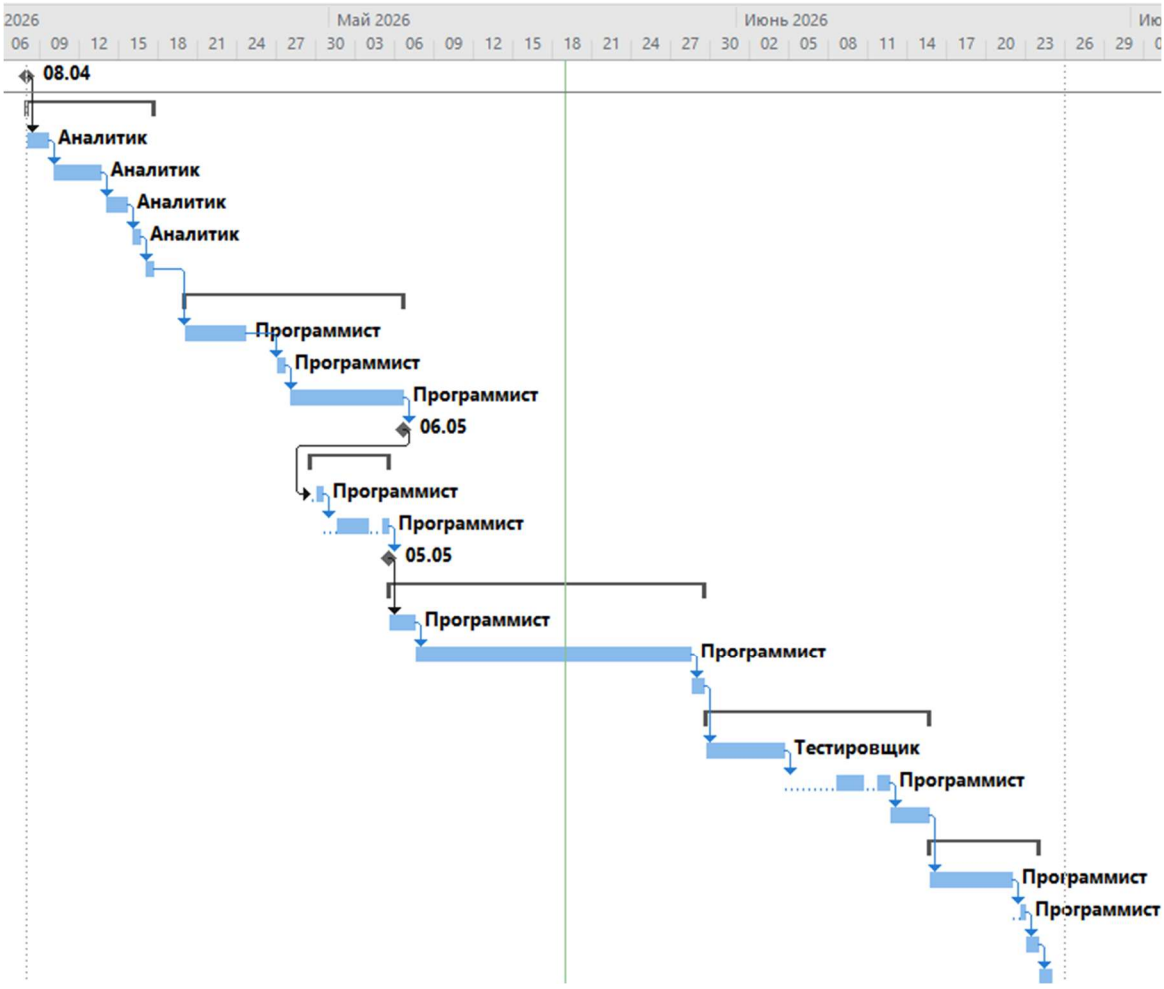

Рисунок 8 – Диаграмма Ганта

В календарный план включено 29 задач, с учетом суммарных этапов и вех. Они охватывают путь от анализа предметной области до сдачи проекта. Основные этапы проекта:

– анализ предметной области;

– проектирование;

– реализация;

– тестирование;

– оформление документации.

Для выполнения данного проекта выделено 3 роли:

– аналитик (64 чел.ч);

– программист (486,33 чел.ч);

– тестировщик (32 чел.ч).

Суммарные трудозатраты по плану составляют 624 чел.ч.

Для оценки бюджета проекта приняты ориентировочные почасовые ставки, представленные в таблице 2.

Таблица 2 – Почасовые ставки исполнителей

Исполнитель	Ставка, руб./час
Аналитик	300
Программист	380
Тестировщик	280

Расчет расходов на разработку приведен на рисунке 9.

Название	Названия ресурсов	Затраты	Трудозатраты
Названия ресурсов: Аналитик	**Аналитик**	**19 200,00 руб.**	**64 ч**
2 Глава 1. Анализ предметной области и постановка задачи	Аналитик	19 200,00 руб.	64 ч
Названия ресурсов: Программист	**Программист**	**184 805,27 руб.**	**486,33 ч**
3 Глава 2. Проектирование панели администратора	Программист	**54 720,00 руб.**	144 ч
4 Глава 3. Реализация панели администратора	Программист	**84 485,27 руб.**	222,33 ч
5 Глава 4. Тестирование панели администратора	Программист	**21 280,00 руб.**	56 ч
6 Оформление и защита ВКР	Программист	24 320,00 руб.	64 ч
Названия ресурсов: Тестировщик	**Тестировщик**	**8 960,00 руб.**	**32 ч**
5 Глава 4. Тестирование панели администратора	Тестировщик	**8 960,00 руб.**	32 ч
ВКР: Панель администрирования корпоративного мессенджера		**212 965,27 руб.**	**582,33 ч**

Рисунок 9 – Расчет расходов на разработку

Таким образом, планирование показало, как разработка приложения укладывается в два календарных месяца, и с помощью полученных данных была рассчитана предварительная стоимость разработки. Полученные оценки носят плановый характер и могут уточняться при изменении состава команды или ставок, но служат основой для контроля хода выполнения работы.

2 Проектирование приложения для администрирования корпоративного мессенджера

2.1 Проектирование архитектуры панели администратора

Панель администратора корпоративного мессенджера проектируется как автономное клиентское веб-приложение, которое не содержит собственной бизнес-логики и не является хранилищем данных, а выступает интерфейсом к серверной части системы. Архитектура клиента основывается на Feature-Sliced Design подходе [10] и так же строится по слоям с разделением ответственности.

На уровне представления располагаются страницы, каждая из которых реализует один или несколько сценариев вариантов использования. Страницы не обращаются к сети напрямую в произвольном виде, а используют общий слой доступа к API и механизмы управления состоянием.

Уровень маршрутизации задает структуру навигации: публичные маршруты и защищенную группу маршрутов, в которой, при попытке доступа неавторизованным пользователем, происходит перенаправление на страницу входа.

В слое доступа к данным клиент формирует запросы к базовому URL API, автоматически добавляет заголовки и токен авторизации, разбирает ответы и преобразует ошибки в исключения с понятным текстом. Для операций чтения и изменения данных на страницах используется TanStack Query React, что дает возможность кэшировать списки, отображать индикацию загрузки и инициировать повторные запросы при ошибках, а также изменять кэш при успешных мутациях [11]. Это снижает дублирование кода на страницах и согласуется с нефункциональными требованиями к отзывчивости интерфейса.

2.2 Проектирование пользовательских сценариев

На основе сформулированных функциональных требований и вариантах использования было проведено проектирование пользовательских сценариев. Под пользовательским сценарием в контексте панели администратора

понимается упорядоченная последовательность действий администратора и реакций системы при достижении конкретной цели.

Сценарии группируются по разделам приложения, которые соответствуют административным задачам предметной области. Для демонстрации пользовательских сценариев используется диаграмма деятельности [12]. Диаграмма деятельности – это поведенческая диаграмма, которая графически описывает поток управления и/или поток данных рамках некоторого процесса, алгоритма или бизнес-сценария. Она показывает последовательность выполнения действий, возможные ветвления, параллельные потоки и ответственность исполнителей.

Диаграмма деятельности для сценария создания рабочего пространства представлена на рисунке 10.

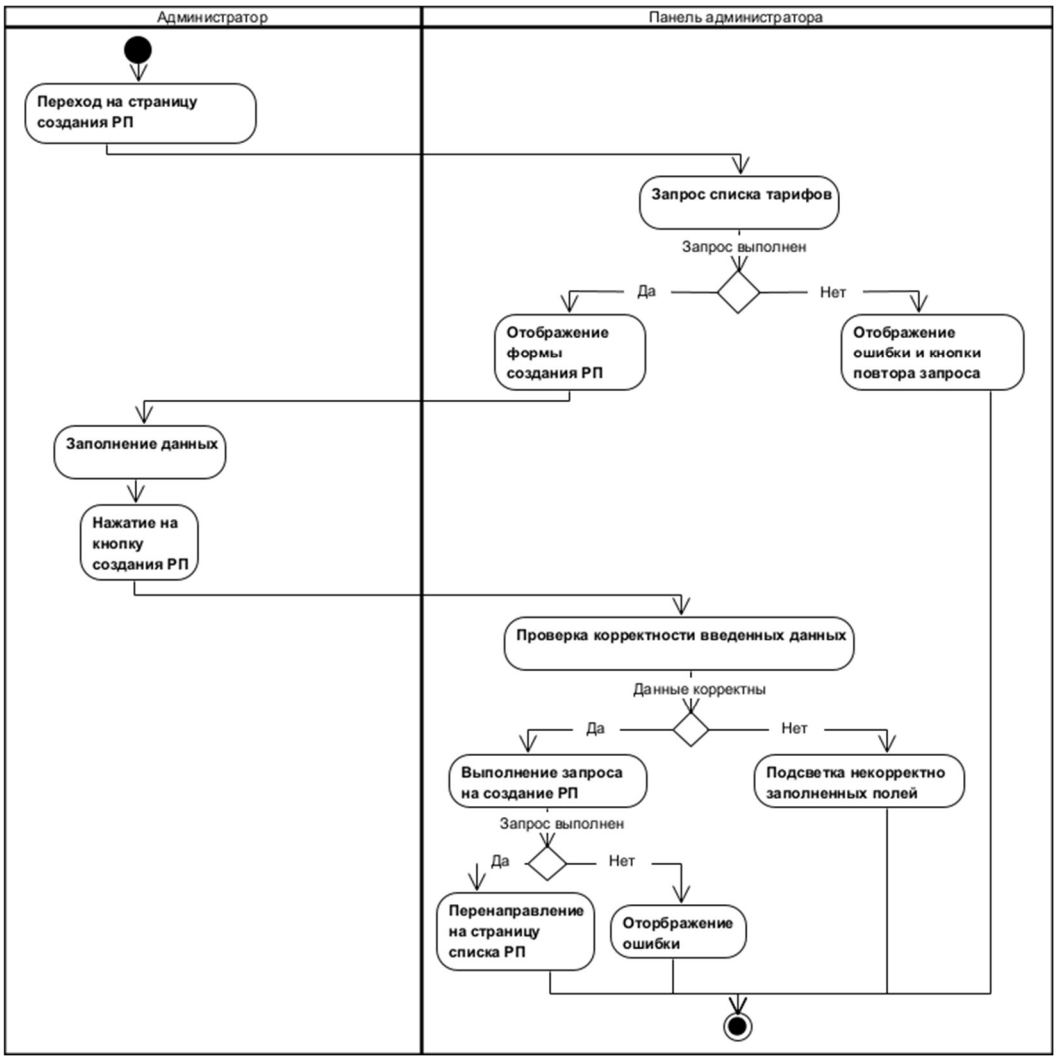

Рисунок 10 – Сценарий создания рабочего пространства

На рисунке 11 представлена диаграмма деятельности для сценария входа в систему.

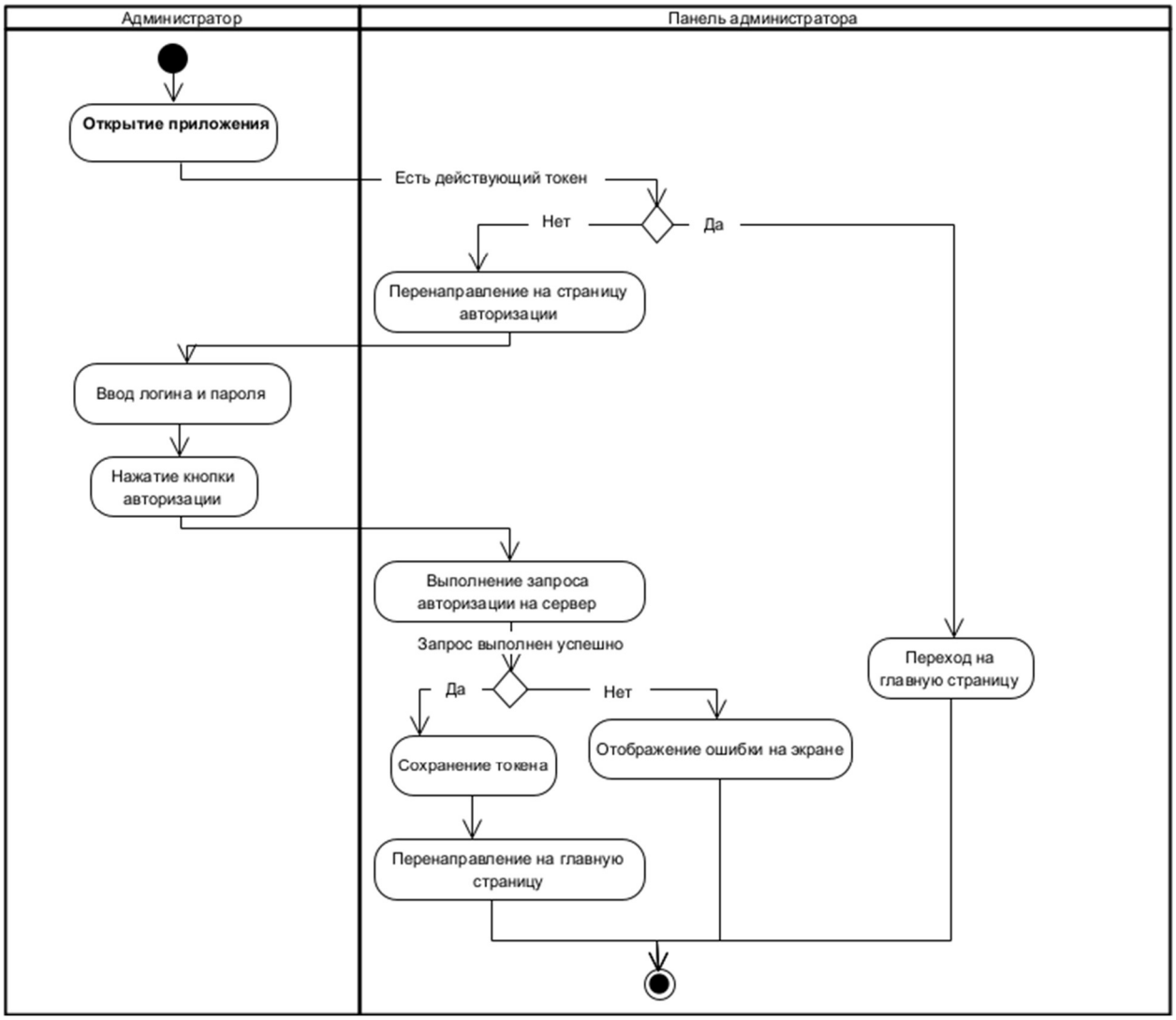

Рисунок 11 – Сценарий авторизации администратора в системе

Данные пользовательские сценарии детально демонстрируют как администратор взаимодействует с системой в рамках часто использующегося функционала.

2.3 Концептуальная модель приложения

Следующим шагом, после рассмотрения пользовательских сценариев, требуется спроектировать набор структур данных, необходимых для работы панели администратора. Для определения таких структур требуется провести разработку концептуальной схемы. Под концептуальной схемой будет

рассматриваться описание предметной области в терминах сущностей, их свойств и связей. Тут будет зафиксировано какие объекты будут храниться в системе, а также их смысловые характеристики и связи. Схема, соответствующая разрабатываемому приложению, приведена на рисунке Б.1 приложения Б.

Для элементов полученной концептуальной схемы приложения были составлены спецификации, приведенные в таблице 3.

Таблица 3 – Спецификация элементов концептуальной схемы

№ п/п	Наименование	Описание
1	Пользователь	Пользователь мессенджера
2	Администратор	Человек, контролирующий работу приложения
3	Рабочее пространство	Область коммуникации внутри одной компании
4	Чат	Отдельная комната для общения в рабочем пространстве
5	Задача	Задача, выделенная в рамках работы одной команды (т. е. в одном чате)
6	Сообщение	Текстовое сообщение внутри чата
7	Жалоба	Описание проблемы в работе приложения, отправляемое администратору
8	Тариф	Условия предоставления организации доступа к мессенджеру

Помимо описания сущностей требуется описать так же и связи между ними. В таблице 4 представлено описание связей сущностей концептуальной модели.

Таблица 4 – Спецификация связей между отношениями концептуальной схемы данных

№ п/п	Сущность 1	Сущность 2	Семантические связи	Мощность	Обязательность
1	Пользователь	Рабочее пространство	В рабочем пространстве находятся пользователи	M : 1	Каждый пользователь должен быть прикреплен к РП; Каждая РП должна иметь пользователей

Продолжение таблицы 4

№ п/п	Сущность 1	Сущность 2	Семантические связи	Мощность	Обязательность
2	Пользователь	Чат	Пользователь создает и управляет своими чатами, а также находится в чатах, куда его пригласили	M : M	В Чате обязательно должен быть хотя бы 1 пользователь - создатель
3	Пользователь	Сообщения	Пользователь отправляет сообщения	1 : M	У сообщения обязательно должен быть автор
4	Пользователь	Задача	Пользователь создает, управляет задачами и выполняет их	1 : M	У задачи обязательно должен быть автор У задачи может не быть исполнителя
7	Администратор	Жалобы	Администратор читает жалобы пользователей	1 : M	-
8	Чаты	Сообщения	Сообщения хранятся в чатах	1 : M	- Сообщение обязательно хранится в конкретном чате
9	Чаты	Задачи	Задачи создаются и хранятся в чатах	1 : M	- Задача обязательно хранится в конкретном чате
10	Чат	Рабочее пространство	Чаты хранятся в рабочем пространстве	M : 1	- Чат обязательно существует только в рамках рабочего пространства
11	Рабочее пространство	Тариф	Тариф регулирует ограничения использования мессенджера	M : 1	- У каждого рабочего пространства должен быть тариф

Таким образом, в результате разработки концептуальной схемы данных панели администратора получен набор отношений, необходимых для дальнейшей разработки приложения. Помимо прочего, полученные данные будут задействованы в следующем шаге – проектировании пользовательского интерфейса.

2.4 Проектирование пользовательского интерфейса

На основе полученных данных о требованиях к разрабатываемому приложению, а также отношениях, принимающих участие в его работе, можно провести проектирование пользовательского интерфейса панели администратора. На данном этапе потребуется определить как человек будет взаимодействовать с системой: какие экраны понадобятся и какие действия на них доступны, а также в каком порядке пользователь достигает нужного действия [13].

Панель администратора – критически важный узел в работе всего корпоративного мессенджера, а значит, что доступ к его функционалу должен строго регулироваться. Следовательно, первой страницей, которую увидит администратор, будет страница авторизации. Здесь реализован базовый функционал для входа в систему: администратору предлагается ввести свой логин и пароль, и выполнить вход с помощью соответствующей кнопки. Проект страницы авторизации представлен на рисунке 12.

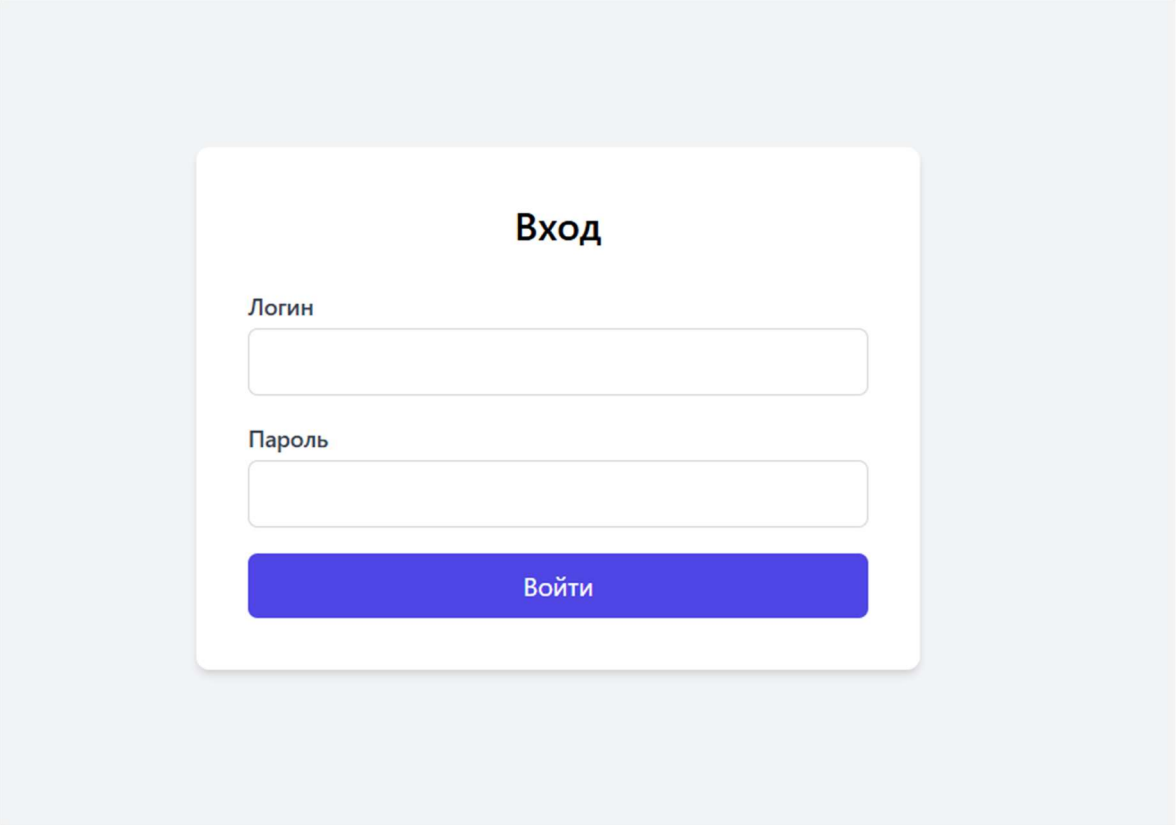

Рисунок 12 – Интерфейс страницы авторизации

После входа в систему, должна открываться главная страница приложения. На этой страница должна отображаться основная информация о мессенджере, а именно: количество пользователей и рабочих пространств в системе, а также жалобы, поступившие от пользователей. Помимо этого, администратору должна быть доступна навигация в другие разделы для работы непосредственно с пользователями, рабочими пространствами и жалобами. Интерфейс главной страницы приведен на рисунке Б.2 в приложении Б.

В разделе «Рабочие пространства» администратору должны быть доступны все необходимые для работы инструменты с рабочими пространствами, а именно: просмотреть детальную информацию, создать новое, удалить и редактировать. Интерфейс, отвечающий этим требованиям, представлен на рисунке Б.3 приложения Б.

Одной из самых важных функций на данной странице является возможность создания новых рабочих пространств. Для этого администратору требуется нажать кнопку «Создать» в верхней части страницы, после чего он будет перенаправлен на страницу создания рабочего пространства. Здесь требуется ввести название, идентификатор руководителя и выбрать тариф для создаваемого пространства. Интерфейс окна создания рабочего пространства представлен на рисунке Б.4 приложения Б.

При выборе нажатии кнопки «Открыть» около рабочего пространства в соответствующем разделе администратор попадет на страницу просмотра детальной информации о выбранном рабочем пространстве. На данной странице будет отображаться основная информация о пространстве, а так же будет возможность ее редактировать, а именно: изменить название, тариф, добавить новых участников, удалить участников, а так же изменить их роли. Так, при добавлении нового участника ему выдаются по умолчанию базовые права, на этой же странице можно выдать ему права руководителя рабочего пространства или же вообще удалить из списка участников данного рабочего пространства. Кроме того, на этой странице должны отображаться даты добавления каждого участника. Интерфейс страницы просмотра и редактирования информации о

рабочем пространстве и его участниках представлен на рисунке Б.5 в приложении Б.

При переходе в раздел «Пользователи» администратору открывается страница с списком всех пользователей корпоративного мессенджера. В списке отображается основная информация об аккаунте, а именно: идентификатор, логин, фамилия, имя и отчество, а также текущий статус. В верхней части страницы располагается панель, с помощью которой возможно выполнить поиск определенных пользователей с использованием соответствующих параметров и фильтров. Интерфейс раздела «Пользователи» представлен на рисунке Б.6 приложения Б.

В разделе «Жалобы» администратор взаимодействует с сообщениями от пользователей по ошибкам или вопросам при работе с корпоративным мессенджером. В правой части страницы находится список отправленных пользователями жалоб, где указывается идентификатор жалобы, логин отправителя и дата создания, помимо этого, присутствуют кнопки просмотра и удаления жалобы. В левой же части страницы находится окно, в котором выводится вся информация по жалобе при ее выборе в правой части экрана. Интерфейс раздела жалоб приведен в рисунке Б.7 в приложении Б.

При переходе в раздел «Настройки и состояние» перед администратором открываются графики, отображающие состояние компонентов системы, данные берутся из платформы для визуализации данных и мониторинга – Grafana, в которую администратор может перейти, нажав кнопку «Открыть Grafana». Интерфейс панели мониторинга показан на рисунке Б.8 приложения Б.

Таким образом, в результате проектирования пользовательского интерфейса панели администратора был получен набор страниц, с помощью которых администратор корпоративного мессенджера будет реализовывать контроль над приложением. На каждом экране были выделены соответствующие элементы, необходимые для выполнения задач в выбранном разделе, а также данные, которые выводятся в таблицах каждого раздела.

3 Реализация панели администратора корпоративного мессенджера

3.1 Реализация общей структуры и архитектуры приложения

Клиент реализован как одностраничное приложение (SPA) на React и TypeScript. Выбор React обусловлен компонентной моделью, удобной для построения форм, таблиц и экранов с частичным обновлением данных без перезагрузки страницы [14]. TypeScript добавляет статическую проверку типов при работе с ответами API и снижает риск ошибок при сопровождении кода [15].

Сборка и локальная разработка выполняются средствами Vite: инструмент обеспечивает быструю сборку при изменениях и формирует оптимизированный статический пакет для размещения на веб-сервере [16]. Стили оформления задаются утилитарными классами Tailwind CSS, что позволяет единообразно описывать отступы, сетки и состояния элементов без отдельного набора CSS-файлов на каждый экран [17].

Код организован по функциональному принципу с выделением общего слоя shared и набора экранов pages:

– pages – маршрутизируемые страницы (вход, обзор, рабочие пространства, пользователи, жалобы);

– shared/layouts – общий каркас с боковым меню и шапкой;

– shared/components – часто используемые элементы;

– shared/api – обращение к API Gateway;

– shared/state – глобальное состояние аутентификации;

– shared/config – параметры окружения.

Точка входа – main.tsx, конфигурация маршрутов сосредоточена в App.tsx. Такое разделение отделяет инфраструктурный код от экранной логики и упрощает добавление новых разделов панели без затрагивания всего приложения. Конфигурация окружения вынесена в отдельный артефакт, поскольку от неё зависит, куда клиент отправляет запросы. Это позволяет

запускать один и тот же код в разных средах: локально, на демонстрационном стенде или в промышленной конфигурации.

Для визуализации структуры панели администратора будет использоваться диаграмма развертывания. Данный тип диаграммы используется для описания того, на каких физических или виртуальных узлах выполняются части системы, какие программные артефакты размещены на этих узлах и по каким каналам они взаимодействуют. Диаграмма компонентов панели администратора представлена на рисунке 13.

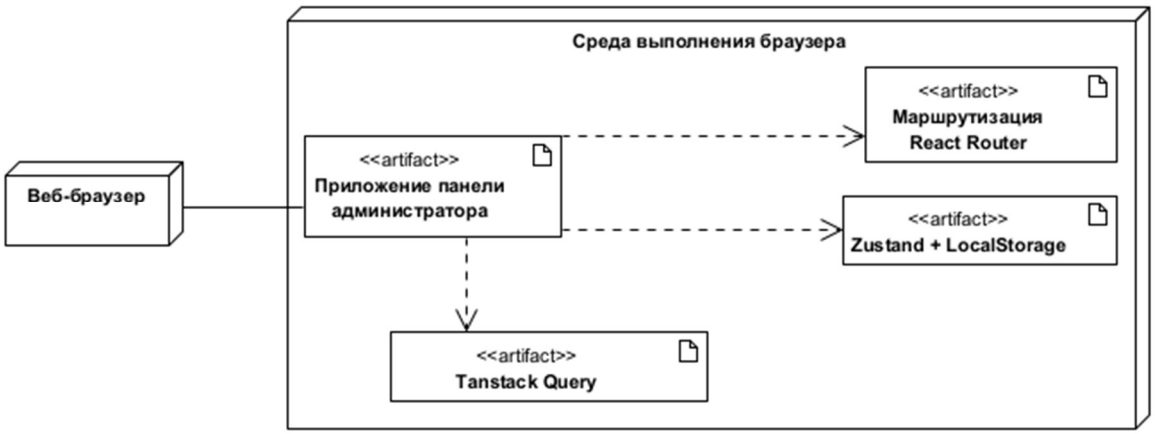

Рисунок 13 – Диаграмма развертывания панели администратора

В разрабатываемой системе можно выделить два основных контура: рабочее место администратора и серверную инфраструктуру корпоративного мессенджера. На рабочем месте администратора выполняется веб-браузер, в котором открывается панель администратора. Клиентская часть панели реализована как SPA-приложение на React и после сборки представляет собой набор статических файлов: HTML, JavaScript и CSS. В режиме разработки эти файлы обслуживаются dev-сервером Vite, а в эксплуатационном варианте могут быть размещены на веб-сервере.

Панель администратора не обращается напрямую к базе данных и отдельным микросервисам. Все бизнес-запросы отправляются через API-шлюз, который является единой точкой входа в backend-систему.

Для мониторинга задействуются Prometheus, который собирает метрики со всех микросервисов, и Grafana, которая визуализирует их в виде дашбордов. Для встраивания графиков из Grafana в разделе «Настройки и мониторинг» используются iframe.

Таким образом, была спроектирована общая архитектура приложения. Подобранное разделение функционала модулей упрощает сопровождение системы и позволяет независимо от серверной части развивать панель администратора.

3.2 Реализация пользовательского интерфейса

3.2.1 Общий макет и стилизация

Пользовательский интерфейс панели администратора реализован набором React-компонентов, которые образуют единый каркас приложения. После прохождения аутентификации администратор попадает в защищенную часть системы, где для всех разделов используется общий макет AppLayout. В нем выделены три постоянные области: боковая навигация, верхняя панель и рабочая область страницы.

Боковая навигация содержит основные разделы: главная страница, рабочие пространства, пользователи, жалобы, мониторинг. Верхняя панель отображает название системы, логин текущего администратора и кнопку выхода. Содержимое активного раздела подставляется через механизм вложенных маршрутов React Router, поэтому общий каркас не дублируется в каждой странице.

Визуальная часть реализована средствами Tailwind CSS. Для экранов используются повторяющиеся приемы: белые карточки на светло-сером фоне, тонкие рамки, таблицы с фиксированными заголовками, компактные формы, единые состояния кнопок при загрузке и отключении. Такой подход позволил обойтись без отдельной UI-библиотеки и сохранить контроль над внешним видом каждого элемента.

Фрагмент формирования навигации и общего каркаса приведен в листинге 1.

Листинг 1 – Фрагмент общего макета панели администратора

```
const navItems = [
  { to: "/", label: ru.nav.dashboard },
  { to: "/workspaces", label: ru.nav.workspaces },
  { to: "/users", label: ru.nav.users },
  { to: "/complaints", label: ru.nav.complaints },
  { to: "/settings", label: ru.nav.settings }
];

<aside className="w-64 bg-white border-r border-gray-200 flex
flex-col">
  <div className="p-4 text-xl font-semibold border-b border-gray-
200">
    {ru.adminPanel}
  </div>
  <nav className="flex-1 p-2 space-y-1">
    {navItems.map((item) => (
      <NavLink key={item.to} to={item.to}>
        {item.label}
      </NavLink>
    ))}
  </nav>
</aside>
```

Подобранная архитектура обеспечивает наиболее удобное расширение функционала при дальнейшем развитии системы без сильных изменений уже существующей кодовой базы.

3.2.2 Раздел управления рабочими пространствами

Раздел рабочих пространств является одним из центральных в панели, так как именно через него администратор создает и сопровождает организационные области корпоративного мессенджера. Реализация раздела разделена на три страницы: список, создание и карточка рабочего пространства.

Страница WorkspacesPage загружает список рабочих пространств через «GET /api/v1/workspaces/all». В запрос передаются параметры «limit», «offset» и при необходимости «tariff_id». Ответ отображается в таблице, где для каждой записи выводятся название, тариф, количество участников и дата создания. Администратору доступны обновление списка, переход к форме создания и

открытие карточки конкретного рабочего пространства. Для длинных списков используются параметры смещения и лимита, что позволяет не загружать весь набор данных сразу.

На странице WorkspaceCreatePage реализована форма создания рабочих пространств. Перед показом формы клиент запрашивает доступные тарифы через «GET /api/v1/workspaces/tariffs», сортирует их по названию и выводит в выпадающем списке. Администратор заполняет название пространства, идентификатор пользователя-руководителя и тариф. Перед отправкой выполняется клиентская проверка: название не должно быть пустым, идентификатор руководителя и тарифа должны быть положительными числами. Если проверка пройдена, вызывается «POST /api/v1/workspaces». В листинге 2 показан фрагмент формы создания, где объединены загрузка тарифов, мутация и проверка данных.

Листинг 2 – Создание рабочего пространства

```
const createWorkspace = useMutation<WorkspaceResponse, Error,
CreateWorkspaceRequest>
(
{
  mutationFn: (payload) =>
    apiFetch<WorkspaceResponse>("/api/v1/workspaces", {
      method: "POST",
      body: payload
    }),
  onSuccess: () => {
    setMessage(ru.workspaceCreate.success);
    setTimeout(() => navigate("/workspaces"), 500);
  }
});

if (!form.name.trim())
{
  setErrorMessage(ru.workspaceCreate.nameRequired);
  return;
}
```

Страница WorkspaceDetailPage предназначена для просмотра и изменения уже созданного рабочего пространства. Верхний блок содержит основные сведения: название, создателя, дату создания и текущий тариф. Ниже располагается форма редактирования названия и тарифа. Отдельный блок

участников размещен под основной информацией: в нем можно добавить пользователя по идентификатору, выбрать роль, изменить роль существующего участника или удалить его из пространства. Такое расположение блоков делает сценарий последовательным: сначала администратор видит параметры рабочего пространства, затем переходит к управлению составом участников.

Для изменения данных используется набор мутаций: обновление рабочих пространств, добавление участника, изменение роли и удаление участника. После успешной операции инвалидируется соответствующий ключ TanStack Query, поэтому таблица обновляется без полной перезагрузки страницы.

3.2.3 Раздел обработки жалоб пользователей

Раздел жалоб реализует сценарий просмотра обращений пользователей и изменения их статуса.

Экран ComplaintsPage построен как рабочая область из двух частей: слева отображается список жалоб, справа – карточка выбранной записи. Такая компоновка позволяет не покидать список при просмотре деталей и быстрее переходить между обращениями.

Список жалоб загружается запросом «GET /api/v1/complaints». В интерфейсе предусмотрен фильтр статуса: все, ожидает, в работе, решена, отклонена. В API для этих состояний используются коды «pending», «in_progress», «resolved» и «rejected» соответственно, а на экране они отображаются русскими подписями через функцию «complaintStatusLabel». Это сохраняет стабильный контракт с сервером и одновременно делает интерфейс понятным администратору.

При выборе записи выполняется запрос «GET /api/v1/complaints/{id}». В карточке отображаются автор, текущий статус, дата создания, описание устройства, текст жалобы и история изменения статусов. Администратор может выбрать новый статус, добавить комментарий и отправить изменение через «PUT /api/v1/complaints/{id}/status».

Фрагмент загрузки списка, деталей и смены статуса приведен в листинге 3.

Листинг 3 – Загрузка и обновление жалоб

```
const complaintsQuery = useQuery<ComplaintsResponse>({
  queryKey: ["complaints", statusFilter],
  queryFn: () => {
    const params = new URLSearchParams({ limit: "20", offset: "0"
});
    if (statusFilter !== "all") params.set("status",
statusFilter);
    return
apiFetch<ComplaintsResponse>(`/api/v1/complaints?${params}`);
  }
});

const updateMutation = useMutation({
  mutationFn: (payload: { id: number; status: string; comment?:
string }) =>
    apiFetch<void>(`/api/v1/complaints/${payload.id}/status`, {
      method: "PUT",
      body: { status: payload.status, comment: payload.comment }
    })
});

const detailQuery = useQuery<ComplaintDetail>({
  queryKey: ["complaint", selectedId],
  queryFn: () =>
apiFetch<ComplaintDetail>(`/api/v1/complaints/${selectedId!}`),
  enabled: selectedId !== null
});
```

Удаление жалобы реализовано через «DELETE /api/v1/complaints/{id}». Во время выполнения операции кнопка показывает состояние «Удаление…», что предотвращает повторное нажатие и дает пользователю обратную связь.

3.2.4 Вспомогательные экраны

Помимо ключевых разделов рабочих пространств и жалоб, в панели реализованы экраны, поддерживающие повседневную работу администратора.

DashboardPage открывается по корневому маршруту. Он выполняет проверку доступности шлюза через «GET /health», отображает краткие показатели по жалобам и рабочим пространствам, а также список последних жалоб. Экран выполняет роль стартовой сводки: администратор сразу видит, отвечает ли серверная часть и есть ли новые обращения.

UsersPage предназначена для просмотра пользователей мессенджера. На странице реализованы поиск по логину, имени и фамилии, фильтр по статусу

присутствия, фильтр по рабочему пространству и постраничная навигация. Такой экран нужен администратору при проверке пользователей, назначении руководителя рабочего пространства и анализе авторов жалоб.

SettingsPage используется для отображения мониторинга. Страница проверяет доступность Grafana по «/api/health», после чего встраивает общий дашборд и отдельные панели через iframe.

3.3 Организация обмена данными с серверной частью

Связь панели администратора с серверной частью построена через REST API. Клиент не обращается к микросервисам напрямую: все запросы отправляются на API-шлюз, адрес которого строго задается переменной окружения VITE_API_BASE_URL.

Для выполнения HTTP-запросов создан общий модуль «admin/src/shared/api/client.ts». Его задача – убрать повторяющийся код из страниц и централизовать обработку типовых ситуаций:

- добавление токена для защищенных запросов;
- установка «Content-Type: application/json»;
- обработка пустого ответа;
- извлечение сообщения об ошибке из JSON или текстового ответа;
- завершение сеанса при ответе с кодом «401 Unauthorized».

Параметр skipAuth используется для запросов, которым не нужен JWT-токен, например для проверки «/health» на странице обзора. Все остальные операции панели выполняются с access-токеном администратора.

Для отдельных предметных областей поверх apiFetch могут создаваться небольшие специализированные функции. Например, поиск пользователей вынесен в «admin/src/shared/api/users.ts»: функция собирает строку параметров и возвращает типизированный ответ. Такой подход удобно применять для ресурсов с несколькими фильтрами.

Работа с серверными данными организована через TanStack Query. Для операций чтения используются useQuery, а для изменения данных – useMutation.

Это позволяет хранить кэш результатов, показывать состояния загрузки и ошибок, а после успешных изменений автоматически помечать связанные данные как устаревшие через invalidateQueries.

Для операций, которые меняют данные на сервере, применяются мутации. Например, при смене статуса жалобы после успешного ответа обновляются как общий список жалоб, так и карточка выбранной жалобы. Благодаря этому администратор сразу видит результат действия в интерфейсе.

Таким образом, в приложении разделены транспортный уровень, управление данными и отображение результата на страницах. Это упрощает расширение панели: при добавлении нового раздела достаточно писать новые запросы и связать их с компонентом страницы, не меняя общий механизм HTTP-обмена.

3.4 Аутентификация и управление состоянием сеанса

3.4.1 Хранение сеанса администратора

Для хранения данных аутентификации используется библиотека Zustand с persist. В хранилище useAuthStore находятся accessToken, refreshToken, expiresAt и объект admin. Данные сохраняются в localStorage под ключом admin-auth, поэтому при обновлении страницы приложение может восстановить состояния сеанса без повторного ввода логина и пароля.

Учетные данные администратора отправляются на отдельный запрос «POST /api/v1/auth/admin/login». В ответ сервер возвращает access-токен, refresh-токен, время жизни access-токена и сведения об администраторе. Метод login сохраняет эти данные в Zustand, а метод logout при наличии refresh-токена отправляет запрос на выход и затем очищает локальное состояние. В листинге 4 описывается хранилище сеанса администратора.

Листинг 4 – Хранилище сеанса администратор

```
export const useAuthStore = create<AuthState>()(
  persist(
    (set, get) => ({
```

```
      accessToken: null,
      refreshToken: null,
      expiresAt: null,
      admin: null,
      isAuthenticated: () => Boolean(get().accessToken),
      login: async (payload) => {
        const res = await
fetch(`${API_BASE_URL}/api/v1/auth/admin/login`, {
          method: "POST",
          headers: { "Content-Type": "application/json" },
          body: JSON.stringify(payload)
        });
        const data = (await res.json()) as LoginResponse;
        set({ accessToken: data.access_token, refreshToken:
data.refresh_token, admin: data.admin });
      }
    }),
    { name: "admin-auth" }
  )
);
```

Таким образом, реализация хранилища сеанса обеспечивает сохранение состояния авторизации пользователя между различными частями административного приложения. Использование отдельного клиентского хранилища позволяет централизованно управлять данными текущего сеанса, хранить сведения об авторизованном администраторе и предоставлять другим компонентам приложения единый способ проверки доступа.

3.4.2 Защита маршрутов

Доступ к разделам панели ограничен компонентом ProtectedRoute. Он проверяет результат isAuthenticated из Zustand-хранилища. Если access-токен отсутствует, пользователь перенаправляется на страницу авторизации, а исходных маршрут сохраняется в параметре state.from. После успешного входа администратор возвращается на страницу, на которую он пытался попасть. Код метода проверки авторизации представлен в листинге 5.

Листинг 5 – Проверка авторизации перед открытие защищенных маршрутов

```
function ProtectedRoute({ children }: Props) {
  const isAuth = useAuthStore((s) => s.isAuthenticated());
  const location = useLocation();
  if (!isAuth) {
    return <Navigate to="/login" replace
```

```
state={{ from: location }} />;
  }

  return <>{children}</>;
}
```

Такой механизм не смешивает проверку доступа с кодом конкретных страниц и другие разделы могут считать, что пользователь уже прошел базовую проверку авторизации.

3.4.3 Серверное и локальное состояние экранов

Для хранения серверных данных используется кэш TanStack Query, так как эти данные имеют свой собственный жизненный цикл: загрузка, фоновое обновление, ошибка, устаревание и повторный запрос.

Так, список жалоб зависит от текущего фильтра статуса, а список рабочих пространств – от лимита, смещения и тарифа. Эти параметры входят в queryKey, поэтому библиотека различает наборы данных и обновляет только нужные фрагменты.

Локальное состояние хранится в самих страницах через useState. В ComplaintsPage это statusFilet, selectedId, updateStatus, updateComment и pendingDeleteId; в WorkspaceDetailPage – формы редактирования рабочих пространств, добавления участника и черновики ролей. Такие значения имеют смысл только в пределах текущего экрана, поэтому переносить их в глобальное хранилище не требуется.

Конфигурационное состояние отделено от пользовательского и серверного. Например, grafanaConfig получает адрес Grafana и параметры дашборда из import.meta.env; если переменные не заданы, применяются значения по умолчанию. Для режима разработки предусмотрен путь «/grafana-proxy», который обслуживается Vite и позволяет встроить графики без ручной настройки CORS.

В итоге состояние панели распределено по назначению: Zustand отвечает за сеанс администратора, TanStack Query – за данные REST API, useState – за временные значения интерфейса, а import.meta.env – за параметры окружения.

Такое разделение делает код проще для сопровождения и снижает риск побочных эффектов между независимыми разделами панели.

Таким образом, в рамках реализации создано SPA-приложение на React и TypeScript, встроенное в архитектуру корпоративного мессенджера через API Gateway. Зафиксированы структура каталогов, общий макет с маршрутизацией, единый HTTP-клиент с поддержкой JWT и экраны для управления рабочими пространствами, жалобами, пользователями и мониторингом. Фрагменты код приложения представлены в приложении Б. Полный код панели администратора приведен в git репозитории [18]. Реализованное приложение внедрено в работу внутри организации. Акт о внедрении представлен в приложении Д.

4 Тестирование приложения администрирования корпоративного мессенджера

4.1 Тестирование методом «белого ящика»

В ходе разработки приложения было проведено тестирование клиентской части методом «белого ящика». Это гарантирует корректность выполнения основных сценариев и операций.

Тестирования белым ящиком предполагает, что тестировщик или разработчик имеет полный доступ к исходному коду, алгоритмам, архитектуре и даже к промежуточным состояниям данных, что позволяет строить тесты не только на основе требований, но и на основе того, как именно система реализует требуемое поведение. Целью данного тестирование является выявление дефектов, связанных с некорректными логическими условиями, неправильной обработкой путей выполнения, недостаточным покрытием операторов, ветвлений или условий, а также с ошибками в управляющих конструкциях, циклах и внутренних интерфейсах между модулями. Такой подход особенно эффективен на ранних этапах разработки, при модульном и интеграционном тестировании, когда критически важно проверить не только то, что функция возвращает верный результат, но и то, что все промежуточные действия выполняются правильно и без побочных эффектов. Основополагающим понятием в тестировании белым ящиком является покрытие кода – метрика, показывающая, какая часть исходного кода была выполнена в процессе прогона тестов.

Таким образом, тестирование белым ящиком позволяет оценить корректность внутренней реализации программного продукта и убедиться, что основные ветви выполнения кода работают ожидаемым образом. В разрабатываемой административной панели этот подход применяется для проверки логики отдельных функций и компонентов, обработки данных, ошибок и состояний интерфейса. Его использование повышает надёжность приложения и облегчает дальнейшее сопровождение кода.

В рамках работы было проведено тестирование белым ящиком метода запроса пользователей из базы данных. Граф выполнения данного метода представлен на рисунке 14.

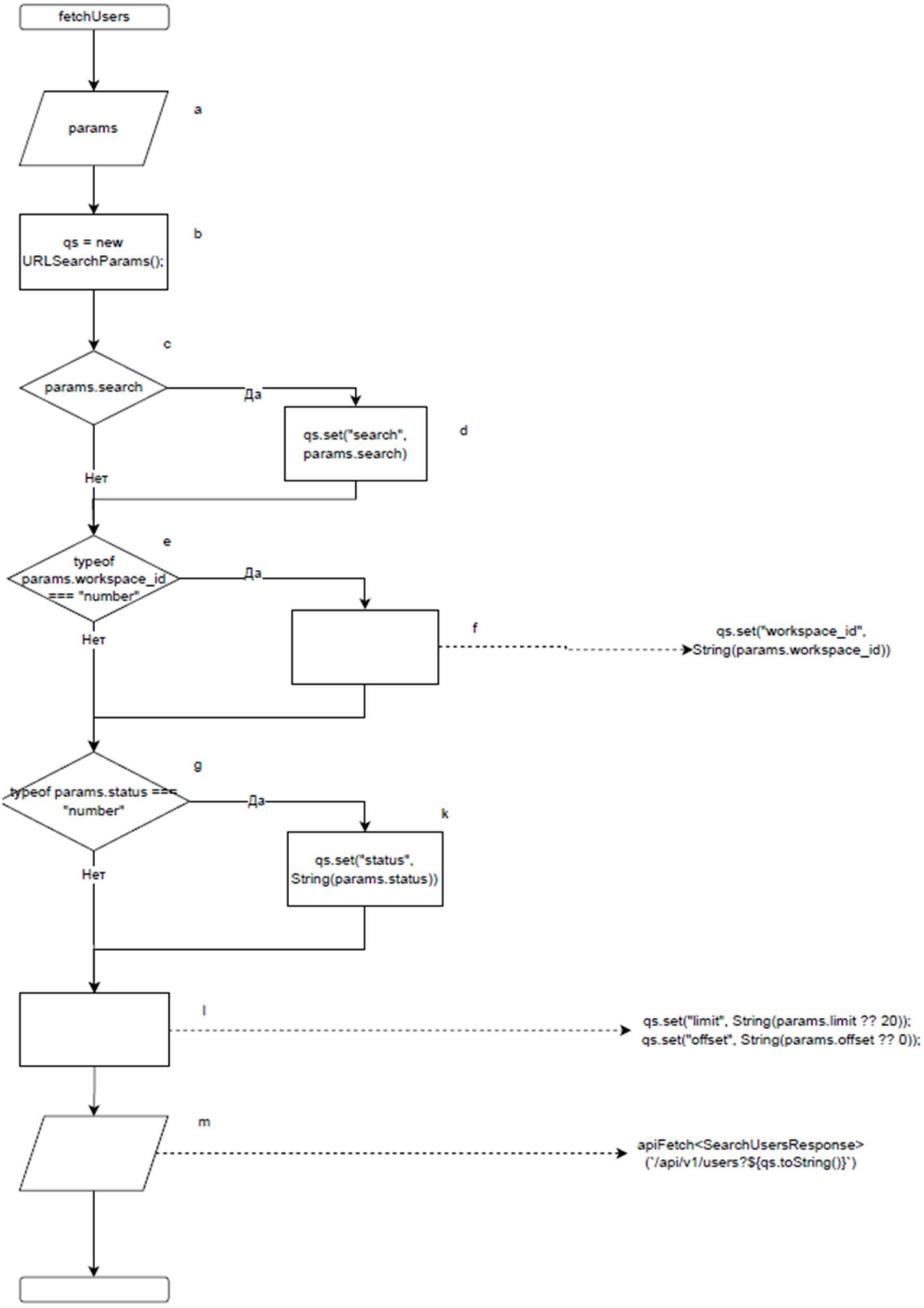

Рисунок 14 – Граф выполнения метода

Данный граф отображает сценарий выполнения метода создания запроса пользователей с сервера. На основе данного графа будет проводиться тестирование.

4.1.1 Покрытие операторов

Критерии покрытия операторов подразумевает выполнение каждого оператора программы по крайней мере один раз. Результаты выполнения покрытия операторов представлены в таблице 5.

Таблица 5 – Результаты покрытия операторов

№	Операторы	Входные данные	Выходные данные	Фактический результат

1	b, d, f, k, l, m	limit: 20
		offset: 0
		search: «user1»
		status: 1
		workspace_id: 1
			limit: 20
			offset: 0
			total: 1
			users: Array(1)
			0: {
			id: 11625
			login: «user11624»
			name: «Имя11624»
			patronymic: «Отчество11624»
			status: 1
			surname: «Фамилия11624»
			}
				Тест пройден. (Рисунок 15)

Ниже, на рисунке 15 приведен результат выполнения теста.

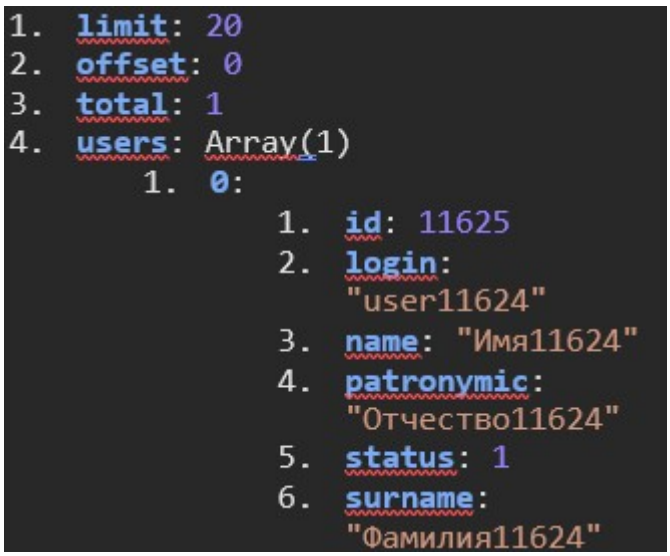

Рисунок 15 – Результаты покрытия операторов

В результате выполнения покрытия был определен сценарий, при котором каждый оператор метода выполняется хотя бы один раз.

4.1.2 Покрытие решений

В соответствии с этим критерием необходимо составить такое число тестов, при которых каждое условие в программе примет как истинное значение, так и ложное значение. Результаты покрытия представлены в таблице 6.

Таблица 6 – Результаты покрытия решений

№	Решения	Входные данные	Выходные данные	Фактический результат
1	c – да e – да g – да	limit: 20 offset: 0 search: «1» status: 1 workspace_id: 1	limit: 20 offset: 0 total: 1 users: [{…}]	Тест пройден. (Рисунок 16)
2	c – нет e – нет g – нет	limit: 20 offset: 0 search: undefined status: undefined workspace_id: undefined	limit: 20 offset: 0 total: 1 users: Array(1)	Тест пройден. (Рисунок 17)

На рисунке 16 представлен результат выполнения теста № 1.

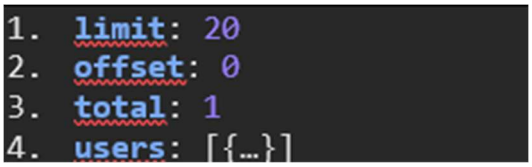

Рисунок 16 – Результат выполнения теста №1

На рисунке 17 представлен результат выполнения теста №2.

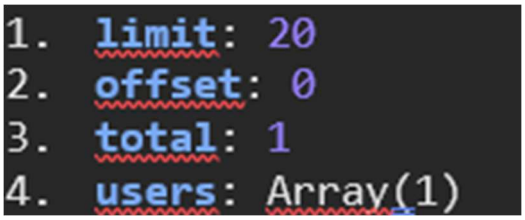

Рисунок 17 – Результат выполнения теста №2

В результате данного этапа были определены сценарии, при которых условия метода выполняются хотя бы один раз.

4.1.3 Комбинаторное покрытие условий

Этот критерий требует, чтобы все возможные комбинации результатов условий в каждом решении, а также каждый оператор выполнились по крайней

мере один раз. Для программ, содержащих только одно условие на каждое решение, минимальный критерий – набор тестов, который вызывает выполнение всех результатов каждого решения по крайней мере один раз и передает управление каждой точке входа программы по крайней мере один раз (чтобы обеспечить выполнение каждого оператора программы по крайней мере один раз). Для программ, содержащих решения, каждое из которых имеет более одного условия, минимальный критерий – набор тестов, вызывающих выполнение всех возможных комбинаций результатов условий в каждом решении и передающих управление каждой точке входа программы по крайней мере один раз. Результаты выполнения этого критерия представлены в таблице 7.

Таблица 7 – Результаты выполнения комбинаторного покрытия

№	Условие	Входные данные	Выходные данные	Фактический результат
1	a, b, c(да), d, e(да), f, g(да), k, l, m	limit: 20 offset: 0 search: «1» status: 1 workspace_id: 1	limit: 20 offset: 0 total: 1 users: [{…}]	Тест пройден. (Рисунок 18.а)
2	a, b, c(да), d, e(нет), g(да), k, l, m	limit: 20 offset: 0 search: «1» status: undefined workspace_id: 1	limit: 20 offset: 0 total: 2 users: (2)	Тест пройден. (Рисунок 18.б)
3	a, b, c(да), d, e(да), f, g(нет), l, m	limit: 20 offset: 0 search: «1» status: 1 workspace_id: undefined	limit: 20 offset: 0 total: 4531 users: (20)	Тест пройден. (Рисунок 18.в)
4	a, b, c(нет), e(да), f, g(да), k, l, m	limit: 20 offset: 0 search: undefined status: 1 workspace_id: 1	limit: 20 offset: 0 total: 1 users: [{…}]	Тест пройден. (Рисунок 18.г)
5	a, b, c(нет), e(нет), g(да), k, l, m	limit: 20 offset: 0 search: undefined status: undefined workspace_id: 1	limit: 20 offset: 0 total: 3 users: (3)	Тест пройден. (Рисунок 18.д)
6	a, b, c(да), d, e(нет), g(нет), l, m	limit: 20 offset: 0 search: «1» status: undefined workspace_id: undefined	limit: 20 offset: 0 total: 13439 users: (20)	Тест пройден. (Рисунок 18.е)

Продолжение таблицы 8

№	Условие	Входные данные	Выходные данные	Фактический результат
7	a, b, c(нет), e(да), f, g(нет), l, m	limit: 20 offset: 0 search: undefined status: 1 workspace_id: undefined	limit: 20 offset: 0 total: 6649 users: (20)	Тест пройден. (Рисунок 18.ж)
8	a, b, c(нет), e(нет), g(нет), l, m	limit: 20 offset: 0 search: undefined status: undefined workspace_id: undefined	limit: 20 offset: 0 total: 20001 users: (20)	Тест пройден. (Рисунок 18.з)

На рисунке 18 представлены результаты выполнения комбинаторного покрытия.

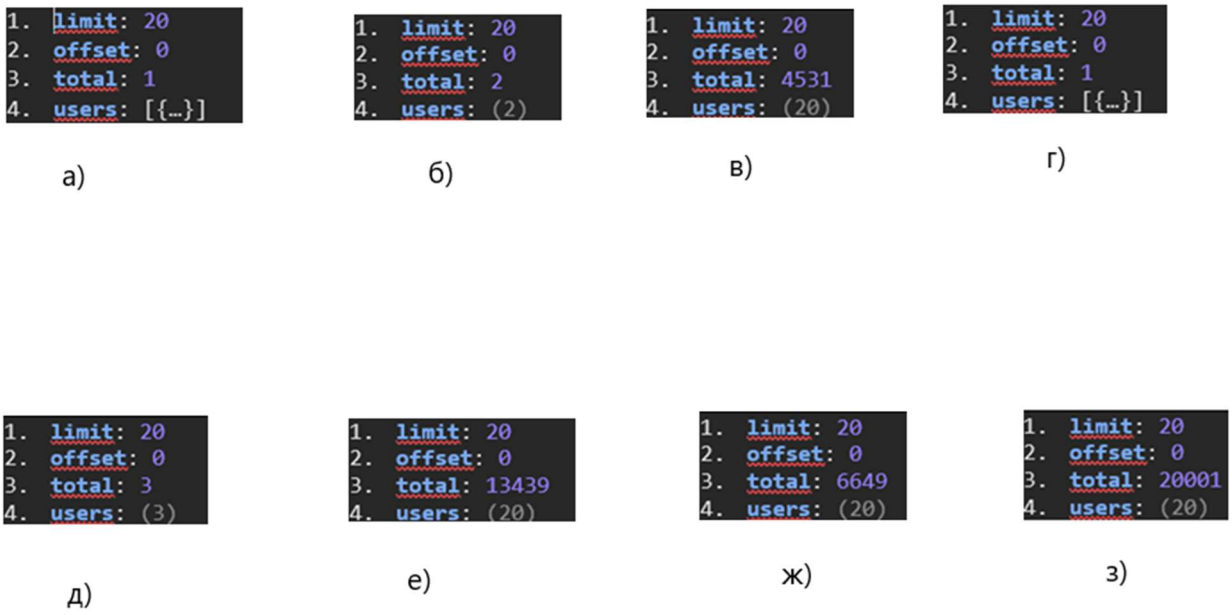

Рисунок 18 – Результаты выполнения тестов

В результате выполнения тестирования была подтверждена корректность работы функции при выполнении любой последовательности операторов и условий.

4.2 Модульное тестирование

Модульное тестирование представляет собой уровень тестирования программного обеспечения, на котором проверяется корректность работы наименьших тестируемых частей приложения – модулей, или юнитов. Основная цель модульного тестирования заключается в изоляции каждого компонента системы и доказательстве того, что он ведет себя в соответствии с проектной

спецификацией независимо от других частей [19]. Преимуществом модульного тестирования является высокая скорость выполнения. Поскольку тестируется небольшой фрагмент кода, такие проверки можно запускать часто: во время разработки, перед фиксацией изменений или в составе автоматизированной проверки проекта. Кроме того, модульные тесты упрощают сопровождение кода, так как фиксируют ожидаемое поведение отдельных функций и помогают понять, какие сценарии уже были учтены при реализации. Такой подход позволяет быстро обнаруживать ошибки при изменении реализации.

Для тестирования метода запроса пользователей были составлены следующие тесты:

– проверяет, что при отсутствии параметров limit и offset в запросе автоматически подставляются значения по умолчанию (20 и 0);

– проверяется, что если передан параметр search, то он правильно добавляется в URL;

– проверяет, что при передаче числа в workspace_id он попадает в строку запроса;

– тестирует, что параметр workspace_id не добавляется, если он отсутствует или имеет тип, отличный от number (например, undefined или строка);

– проверяет, что status добавляется, когда передан как число;

– удостоверяется, что status не включается в URL при отсутствии или нечисловом значении;

– проверяет одновременную передачу всех возможных параметров (search, workspace_id, status, limit, offset) – все они должны оказаться в URL;

– проверяет, что функция возвращает именно тот объект, который вернул apiFetch (т. е. корректно передает результат);

– убеждается, что если apiFetch выбрасывает ошибку, то она пробрасывается дальше, не перехватываясь внутри fetchUsers.

Результаты тестирования представлены на рисунке 19.

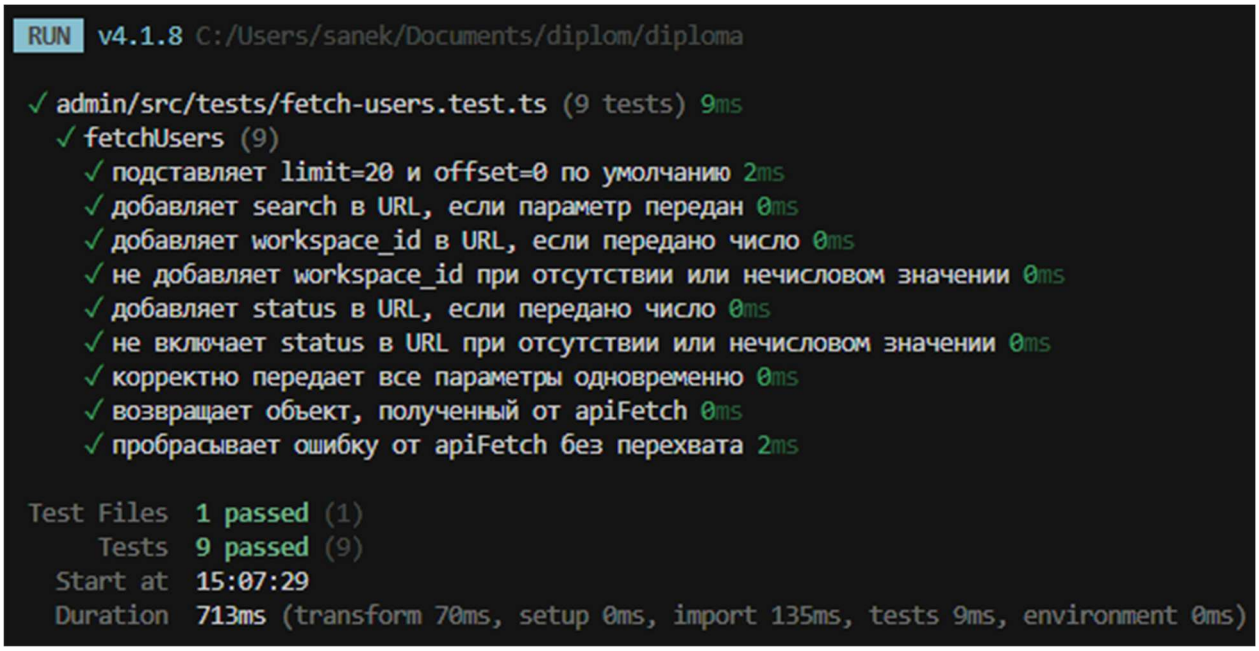

Рисунок 19 – Результаты модульного тестирования

Таким образом, модульное тестирование показала корректность выполнения работы метода запроса пользователей. Код метода модульного тестирования приведен в листинге Г.1 в приложении Г.

4.3 Оценка качества кода

4.3.1 SLOC тестирование

Source Lines of Code (SLOC) – это метрика, показывающая количество строк исходного кода в проекте или модуле. Под этим термином обычно понимают не отдельный вид тестов, а анализ объема кода как дополнительную метрику качества вместе с функциональными и модульными проверками. Сама по себе метрика не показывает качество кода, но используется в связке с другими показателями: покрытием тестами, метриками Холстеда, цикломатической сложностью и результатами функционального тестирования.

Общее количество строк файла – 45, из них пустых строк – 5, а строки с комментариями – 4.

Физические строки кода рассчитываются по формуле: Общее количество строк – количество пустых строк и строк с комментариями: 45 – 5 – 4 = 36.

Логические строки кода – 10.

Таким образом, при размере файла в 45 строк логическая часть программы составляет всего около четверти – 10 строк. При этом соотношение пустых строк и строк с комментариями к общему количеству составляет 20%, что обеспечивает приемлемую читаемость и документированность кода.

4.3.2 Метрики Холстеда

Это набор статических метрик программного кода, предложенный Морисом Холстедом в 1970-х годах. Они оценивают сложность программы по количеству и разнообразию операторов и операндов в исходном тексте, без запуска программы [19]. Для расчета метрик Холстеда был произведен подсчет операторов и операндов модуля запроса пользователей. Результат подсчета операторов представлен на рисунке 20.

Оператор	Количество повторений
тип данных	9 (number) + 5 (string) + 3 (объявленные внутри файла (сам факт объявления не учитывался)) = 17
вызов функции	11
оператор присваивания (=)	2
return	1
скобки ()	16
скобки {}	6
условный оператор if	3
оператор нулевого слияния ??	2
оператор необязательного связывания (имя_параметра?)	9
оператор полного сравнения ===	2
оператор присваивания типа :	16
оператор определения типа typeof	2
Уникальных операторов (n_1)	12
Всего операторов (N_1)	99

Рисунок 20 – Подсчет операторов модуля

Результат подсчета операндов представлен на рисунке 21.

Операнд	Количество повторений
qs	7
params	9
status	4
limit	3
offset	3
workspace_id	3
search	3
result	2
id	1
login	1
surname	1
name	1
patronymic	1
status	1
20	1
0	1
“number”	1
“search”	1
“workspace_id”	1
“limit”	1
“offset”	1
Уникальных операндов (n2)	21
Всего операндов (N2)	47

Рисунок 21 – Подсчет операндов модуля

На основе полученных данных можно произвести расчет метрик. Сначала требуется вычислить размер словаря программы по формуле:

$$n = n_1 + n_2,$$

где n_1 – количество уникальных операторов;

n_2 – количество уникальных операндов.

В разрабатываемой системе 12 уникальных операторов и 21 уникальных операндов. Следовательно, размер словаря программы составляет 33.

Далее требуется вычислить длину программы по формуле:

$$N = N_1 + N_2,$$

где N1 – количество всех операторов;

N2 – количество всех операндов

Исходя из результатов подсчета, по которым в выбранном методе имеется 99 операторов и 47 операндов, можно произвести расчет длины программы, в данном случае равный 146.

Следующим шагом требуется вычислить объем программы (V) по формуле:

$$V = N \times \log_2 \text{n}.$$

Используя полученные значения размера словаря и длины программы, можно установить, что объем программы составляет 736,48.

Помимо этого, требуется вычислить сложность программы (D) по формуле:

$$D = \frac{n_1}{2} \times \frac{N_2}{\text{n}_2}.$$

Подставив значения из результатов подсчета операторов и операндов программы, получаем сложность программы равную 13,429.

Используя полученные данные, можно рассчитать усилие на понимание (E) по формуле:

$$E = V \times D,$$

На основе рассчитанных данных, усилие на понимание оценивается в 9890,19.

По результатам расчетов можно сказать, что сложность программы (D) невысокая, а небольшой объем программы (V) позволяет прочитать и понять код без особых сложностей.

4.3.3 Метрика Чепина

Сущность метрики Чепина заключается в вычислении взвешенной суммы управляющих конструкций, где каждая конструкция получает вес, зависящий от ее типа и уровня вложенности [20]. Данная метрика ориентирована на оценку сложности модуля с точки зрения количества и вложенности управляющих конструкций, что напрямую влияет на удобство тестирования и сопровождения кода.

Для проведения расчетов метрик Чепина используется следующая формула:

$$Q = 1 \times P + 2 \times M + 3 \times C + 0.5 \times T,$$

где P – вводимые переменные;

M – модифицируемые переменные;

C – управляющие переменные;

T – транзитные переменные.

В составе взятой функции присутствует две модифицируемые переменные и пять управляющих. Таким образом, информационная сложность функции fetchUsers составляет 21, что является приемлемым уровнем сложности написанного кода. Основную долю сложности составляют управляющие переменные, что объясняется достаточно большим количеством параметров, передаваемых на вход функции, которые в дальнейшем серьезно влияют на ее работу.

4.3.4 Автоматизированная проверка

Для проведения автоматизированной проверки метрик функции был составлен тест с использованием языковых средств, используемых в основной программе. Автоматизированная проверка метрик используется для объективной оценки качества программной реализации и контроля соответствия кода установленным требованиям. Скрипт анализирует файлы исходного кода, исключает служебные и нерелевантные директории, после чего формирует отчет с рассчитанными значениями. К числу анализируемых показателей можно отнести:

– количество строк кода – отражает объем функции или модуля;

– цикломатическую сложность – показывает число независимых путей выполнения внутри функции;

– когнитивную сложность – характеризует трудность понимания логики функции человеком;

– количество параметров функции – помогает выявлять функции со слишком широким набором входных данных;

– уровень вложенности – показывает глубину условных операторов, циклов и других управляющих конструкций.

В отчете фиксируются сведения о функциях, их расположении в проекте и вычисленных показателях сложности. Это позволяет быстро перейти от общей оценки проекта к конкретным фрагментам кода, которые могут потребовать дополнительного внимания. Код метода для проверки метрик представлен в листинге Г.2 в приложении Г.

Фрагмент результатов тестирования представлен на рисунке 22.


```
[
  {
    "index": 0,
    "name": "App",
    "kind": "function declaration",
    "location": "App.tsx:14:1",
    "paramsCount": 0,
    "hasReturnType": false,
    "bodyLines": 23,
    "cyclomaticComplexity": 1,
    "statementsCount": 1,
    "maxNestingDepth": 0
  },
  {
    "index": 1,
    "name": "<anonymous>",
    "kind": "arrow function",
    "location": "pages/ComplaintsPage.tsx:45:60",
    "paramsCount": 1,
    "hasReturnType": false,
    "bodyLines": 1,
    "cyclomaticComplexity": 1,
    "statementsCount": 0,
    "maxNestingDepth": 0
  },
  {
    "index": 2,
    "name": "formatDate",
    "kind": "function declaration",
    "location": "pages/ComplaintsPage.tsx:47:1",
    "paramsCount": 1,
    "hasReturnType": false,
    "bodyLines": 12,
    "cyclomaticComplexity": 2,
    "statementsCount": 2,
    "maxNestingDepth": 1
  },
  {
```


Рисунок 22 – Результаты автоматизированной проверки

В результате проведения нескольких видов тестирования и расчета характеристик программного кода было получено наиболее исчерпывающее подтверждение корректности работы программы и высокого качества написанного кода.

Заключение

В ходе выполнения выпускной квалификационной работы разработано веб-приложение для администрирования корпоративного мессенджера. Цель работы заключалась в создании клиентской панели, позволяющей администратору выполнять основные операции управления системой: проходить аутентификацию, просматривать пользователей, создавать и настраивать рабочие пространства, обрабатывать жалобы и контролировать состояние сервисов.

На этапе анализа предметной области рассмотрены особенности администрирования корпоративных коммуникационных систем, определены основные участники и сущности приложения. Выделены функции администратора, пользователя и руководителя рабочего пространства, а также сформированы требования к панели администратора. Дополнительно выполнен сравнительный анализ существующих решений, что позволило определить важные критерии для разрабатываемой системы: удобство навигации, безопасность доступа, возможность централизованного управления рабочими пространствами и наличие инструментов контроля состояния приложения.

На этапе проектирования определена архитектура решения и место панели администратора в составе корпоративного мессенджера. Клиентская часть спроектирована как одностраничное веб-приложение, взаимодействующее с серверной инфраструктурой через API Gateway. Описаны пользовательские сценарии, подготовлены диаграммы деятельности, разработана концептуальная модель данных и определены основные сущности предметной области: пользователь, администратор, рабочее пространство, участник рабочего пространства, тариф, жалоба и история статусов, а также составлена CRUD-матрица, связывающая сущности с допустимыми операциями для разных ролей.

Практическая реализация выполнена с использованием React, TypeScript, Vite, Tailwind CSS, TanStack Query и Zustand. В приложении реализован общий макет панели администратора с боковой навигацией, защищенными маршрутами

и русифицированным интерфейсом. Для работы с сервером создан единый API-клиент, добавляющий JWT-токен к защищенным запросам и централизованно обрабатывающий ошибки. Состояние сеанса администратора хранится в Zustand с сохранением в localStorage, а серверные данные загружаются и обновляются через TanStack Query.

В результате реализованы ключевые разделы панели администратора. Раздел рабочих пространств позволяет просматривать список рабочих пространств создавать новые пространства, изменять параметры существующих записей и управлять участниками. Раздел жалоб обеспечивает просмотр обращений пользователей, фильтрацию по статусу, загрузку подробной информации, смену статуса и удаление записей. Раздел пользователей предоставляет поиск и фильтрацию участников системы. Раздел «Настройки и состояние» отображает графики мониторинга Grafana, что позволяет оценивать доступность и нагрузку сервисов.

Разработанное приложение соответствует поставленным функциональным требованиям и поддерживает основные сценарии работы администратора. Панель не дублирует бизнес-логику микросервисов и не обращается напрямую к базе данных: все операции выполняются через API Gateway, что сохраняет разделение ответственности между клиентской и серверной частями. Такой подход упрощает сопровождение системы и позволяет независимо развивать интерфейс администрирования. Помимо этого, было проведено внедрение панели администратора в составе корпоративного мессенджера.

Список использованных источников

1. Юсупова, А.С. Влияние цифровой трансформации на внутренние коммуникации организаций / А.С. Юсупова, Г.Г. Мингазизова, Э.И. Басырова, Э.Б. Гаязова // Финансовый менеджмент. – 2024. – №12. – С. 216 – 225.

2. Куловерова, О. М. Особенности языка электронной коммуникации: на примере чатов немецкоязычных сайтов / О.М. Куловерова, О. Н. Русецкая // Вестник Амурского государственного университета. Серия: Гуманитарные науки. – 2014. – №64. – С.183 – 188.

3. Руднева Е. Тариф // Банки.ру [Электронный ресурс]. URL: https://www.banki.ru/wikibank/tarif/?ysclid=mp9m6kwl67453168994 (дата обращения: 04.05.2026).

4. Mattermost: что за мессенджер и как им пользоваться // РБК Тренды [Электронный ресурс]. URL: https://trends.rbc.ru/trends/social/69ce5f5d9a79472160693592?ysclid=mp9thci3du361371856 (дата обращения: 06.05.2026).

5. VK Teams: обзор российского мессенджера // Compass [Электронный ресурс]. URL: https://getcompass.ru/blog/posts/vk-teams?ysclid=mpd55rmvrg708393923 (дата обращения: 06.05.2026).

6. Rocket.Chat // Hostkey [Электронный ресурс]. URL: https://hostkey.ru/documentation/marketplace/communication/rocketchat/?ysclid=mp9vdlz7lx658153477 (дата обращения: 07.06.2025).

7. Джим Арлоу. UML2 и Унифицированный процесс. Практический объектно ориентированный анализ и проектирование/Джим Арлоу, Айла Нейштадт. – Санкт Петербург, Издательство Символ-Плюс, 2007. – 624с.

8. Best Web Development Frameworks to Use in 2026 // Cogent university [Электронный ресурс]. URL: https://www.cogentuniversity.com/post/best-web-development-frameworks-to-use-in-2026 (дата обращения: 15.05.2026)

9. Тебайкина, Н.И. Управление проектами в MS Project / Н.И. Тебайкина. – Екатеринбург, Издательство УГТУ-УПИ, 2009. – 206с.

10. Feature-Sliced Design: Как FSD спасает от хаоса в растущих проектах // Potapov.me [Электронный ресурс]. URL: https://potapov.me/ru/make/fsd-architecture?ysclid=mpr9jiqw7q729053872 (дата обращения: 29.05.2026).

11. Overview // Tanstack Query [Электронный ресурс]. URL: https://tanstack.com/query/latest/docs/framework/react/overview (дата обращения: 30.05.2026).

12. Диаграмма активностей (Activity diagram) // Ensi [Электронный ресурс]. URL: https://docs.ensi.tech/analyst-guides/tools/diagrams/uml/activity-diagram?ysclid=mpscn4bxo554977592 (дата обращения: 01.06.2026).

13. Купер А. Интерфейс. Основы проектирования взаимодействия / А. Купер – Санкт Петербург, Издательство Питер, 2021 – 720с.

14. Начало работы // React [Электронный ресурс]. URL: https://ru.react.js.org/docs/getting-started.html (дата обращения: 01.06.2026).

15. Введение в TypeScript // metanit.com [Электронный ресурс]. URL: https://metanit.com/web/typescript/1.1.php (дата обращения: 01.06.2026).

16. Vite + React: настройка, HMR, .env и секреты быстрой разработки // /dev/notes [Электронный ресурс]. URL: https://www.dev-notes.ru/articles/react/guide-to-using-vite-with-react/?ysclid=mq46fhy1tc819923965 (дата обращения: 02.06.2026).

17. Обзор синтаксиса Tailwind CSS // Хабр [Электронный ресурс]. URL: https://habr.com/ru/companies/otus/articles/796101/ (дата обращения: 03.06.2026).

18. Корпоративный мессенджер // Github [Электронный ресурс]. URL: https://github.com/niksahn/diploma (дата обращения: 03.06.2026).

19. Хориков В. Принципы юнит-тестирования / В. Хориков – Санкт-Петербург, Издательство Питер, 2021.

20. Белик А.Г., Цыганенко В.Н. Качество и надежность программных средств / А.Г. Белик, В.Н. Цыганенко – Омск, Издательство ОмГТУ, 2018. – 80 с.

21. Программный код и его метрики // Хабр [Электронный ресурс]. URL: https://habr.ru (дата обращения: 04.06.2026).

Приложение А

(справочное)

Глоссарий

Корпоративный мессенджер – Информационная система для делового взаимодействия внутри организации, включающая обмен сообщениями, рабочие пространства, задачи, пользователей и обращения.

Панель администратора – Веб-приложение для выполнения административных функций: входа администратора, просмотра пользователей, создания и настройки рабочих пространств, обработки жалоб и контроля состояния системы.

Администратор – Пользователь с расширенными правами, работающий в панели администратора и выполняющий управление отдельными сущностями корпоративного мессенджера.

Пользователь – Участник корпоративного мессенджера, который работает в рабочих пространствах, участвует в чатах, выполняет задачи и может отправлять жалобы.

Рабочее пространство – Логически отделенная область мессенджера, объединяющая пользователей, чаты, задачи и настройки конкретной организации, команды или подразделения.

Руководитель рабочего пространства – Пользователь, назначенный ответственным за рабочее пространство и обладающий расширенными правами внутри него.

Участник рабочего пространства – Пользователь, добавленный в конкретное рабочее пространство с определенной ролью и датой вступления.

Тариф –Набор условий или ограничений, назначаемый рабочему пространству и определяющий параметры его использования.

Жалоба пользователя – Обращение пользователя, содержащее описание проблемы, сведения об устройстве, дату создания и текущий статус обработки.

Статус жалобы – Состояние обработки обращения: ожидает рассмотрения, находится в работе, решено или отклонено.

API Gateway (API шлюз) – Единая точка входа в серверную часть системы, через которую клиентские приложения отправляют REST-запросы к микросервисам.

React – Библиотека для разработки пользовательских интерфейсов на основе компонентов. В проекте используется для реализации страниц панели администратора.

TypeScript – Расширение JavaScript со статической типизацией, применяемое для описания структур запросов, ответов API и состояния приложения.

Vite – Инструмент сборки и локальной разработки frontend-приложения, обеспечивающий быстрый запуск и формирование статического бандла.

Tailwind CSS – Утилитарный CSS-фреймворк, используемый для стилизации карточек, таблиц, форм, кнопок и общей сетки интерфейса.

TanStack Query – Библиотека для загрузки, кэширования, обновления и инвалидации серверных данных в React-приложении.

Zustand – Библиотека управления состоянием, используемая для хранения данных сеанса администратора: токенов и информации о профиле.

Prometheus – Система сбора метрик, получающая данные о состоянии сервисов через специальные endpoint-ы мониторинга.

Grafana – Средство визуализации метрик, используемое для отображения графиков состояния системы в разделе «Настройки и состояние».

Приложение Б

(рекомендуемое)

Графическая часть

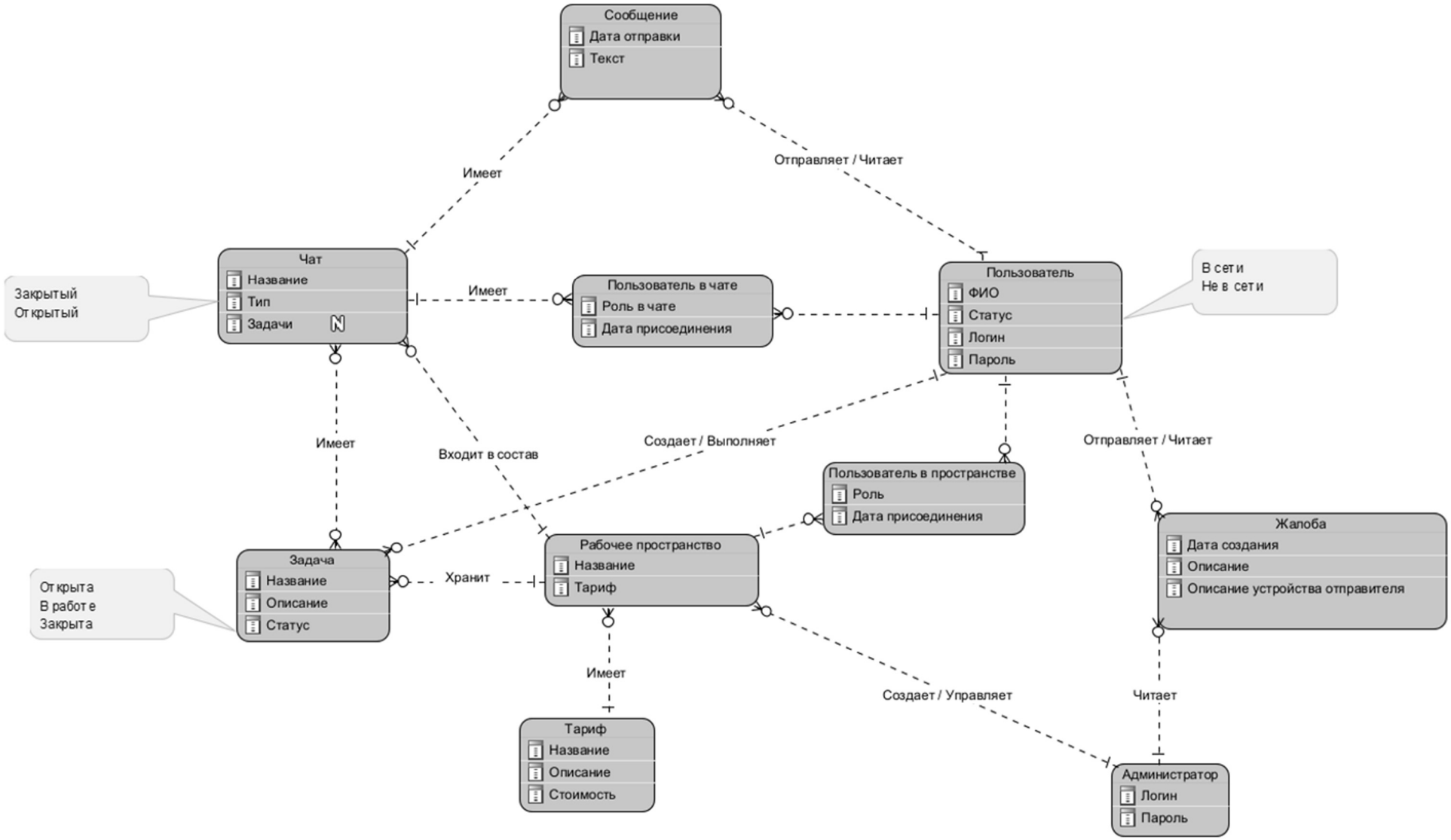

Рисунок Б.1 – Концептуальная схема приложения

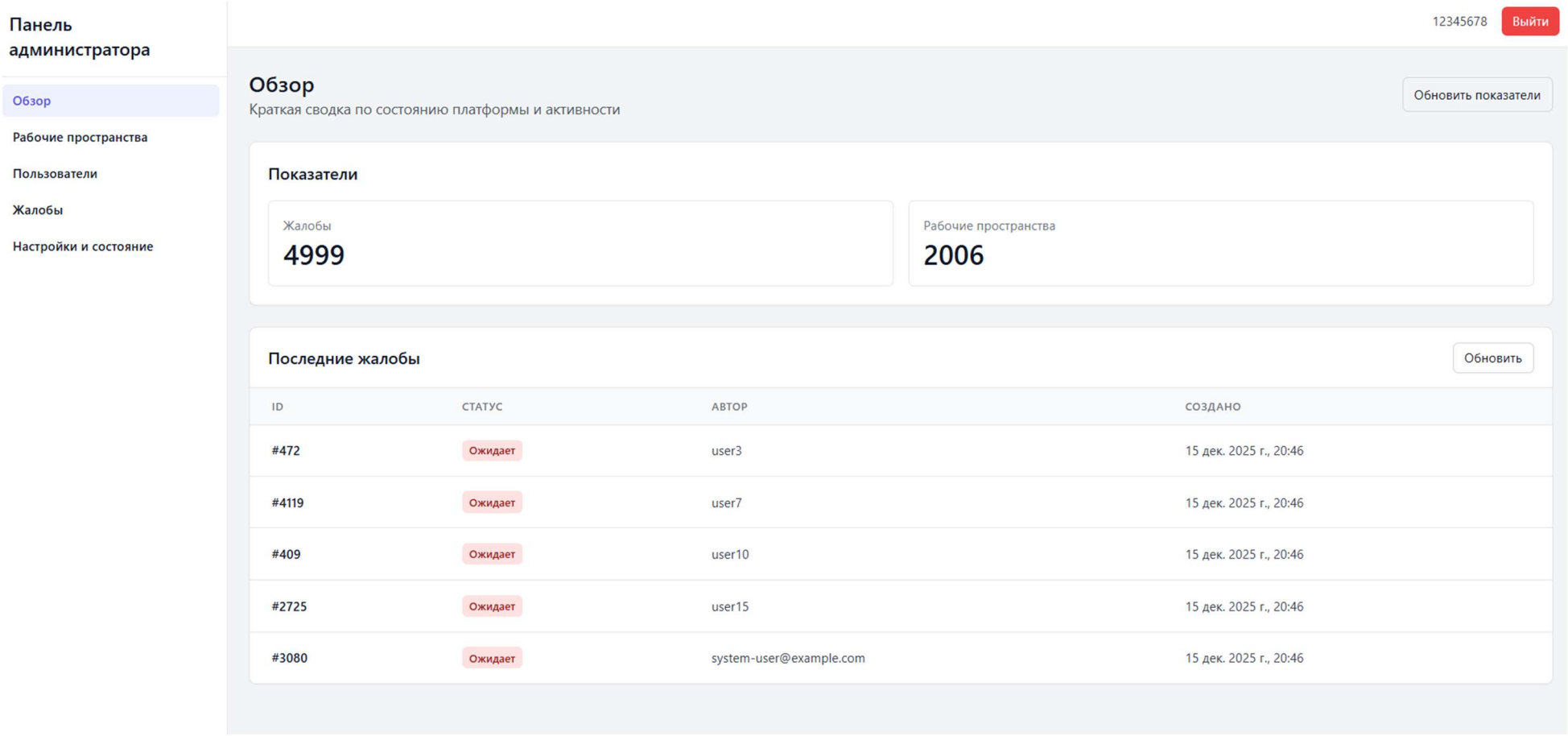

Рисунок Б.2 – Интерфейс главной страницы

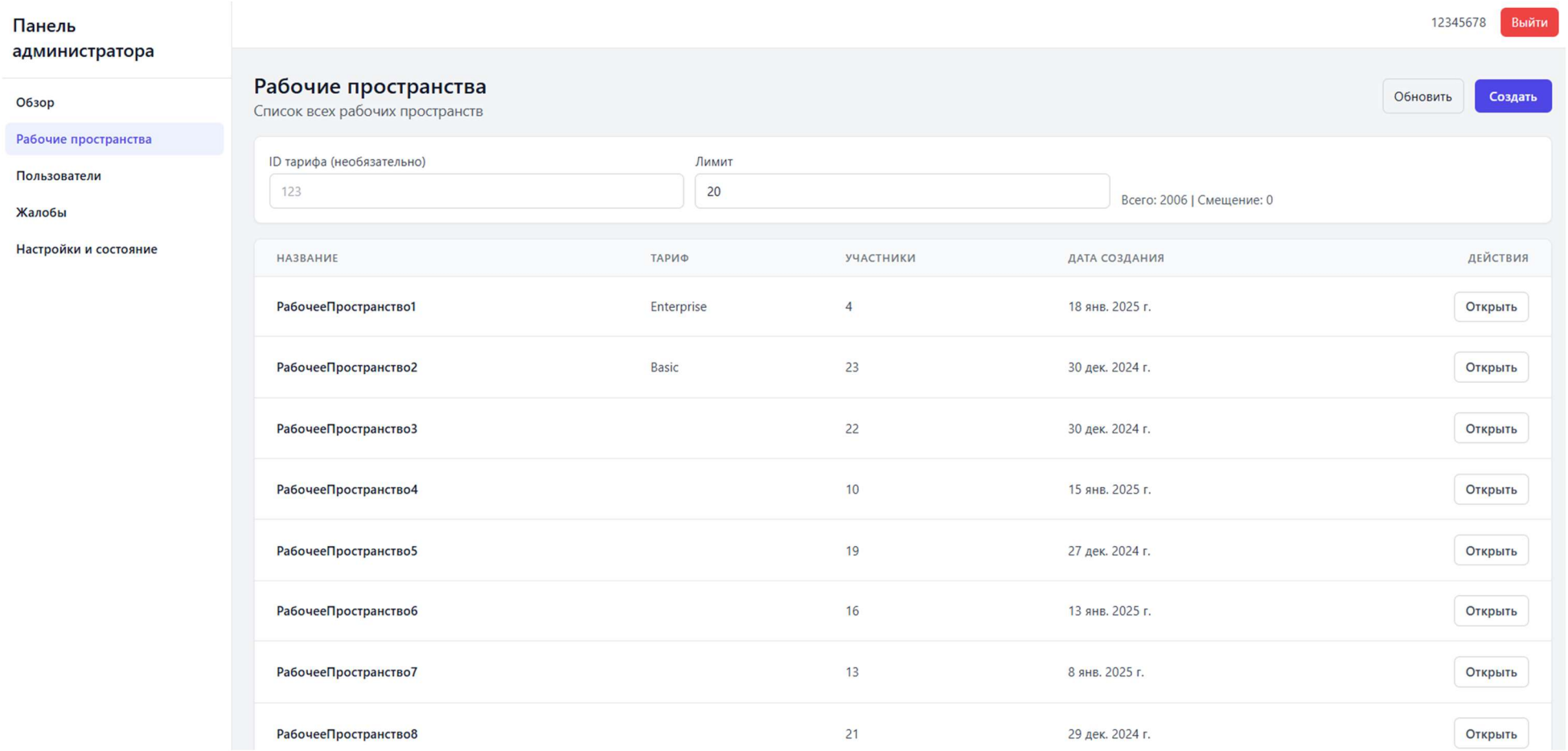

Рисунок Б.3 – Интерфейс раздела рабочих пространств

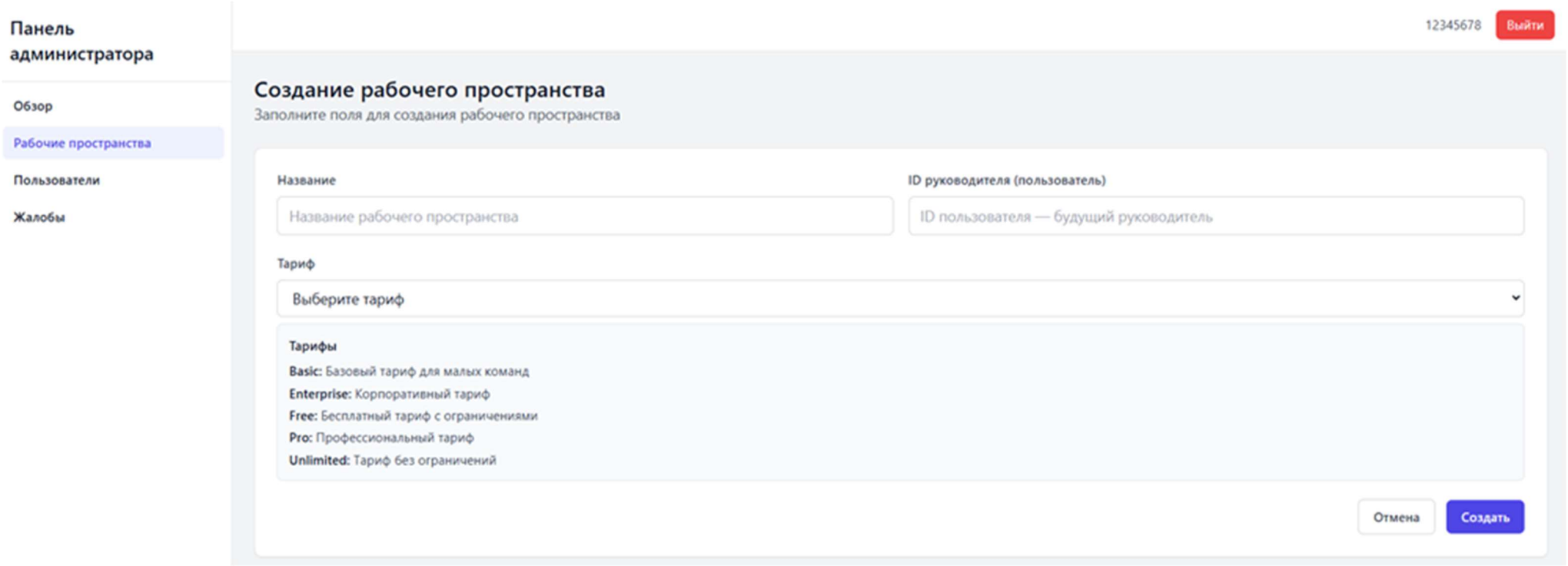

Рисунок Б.4 – Интерфейс окна создания рабочего пространства

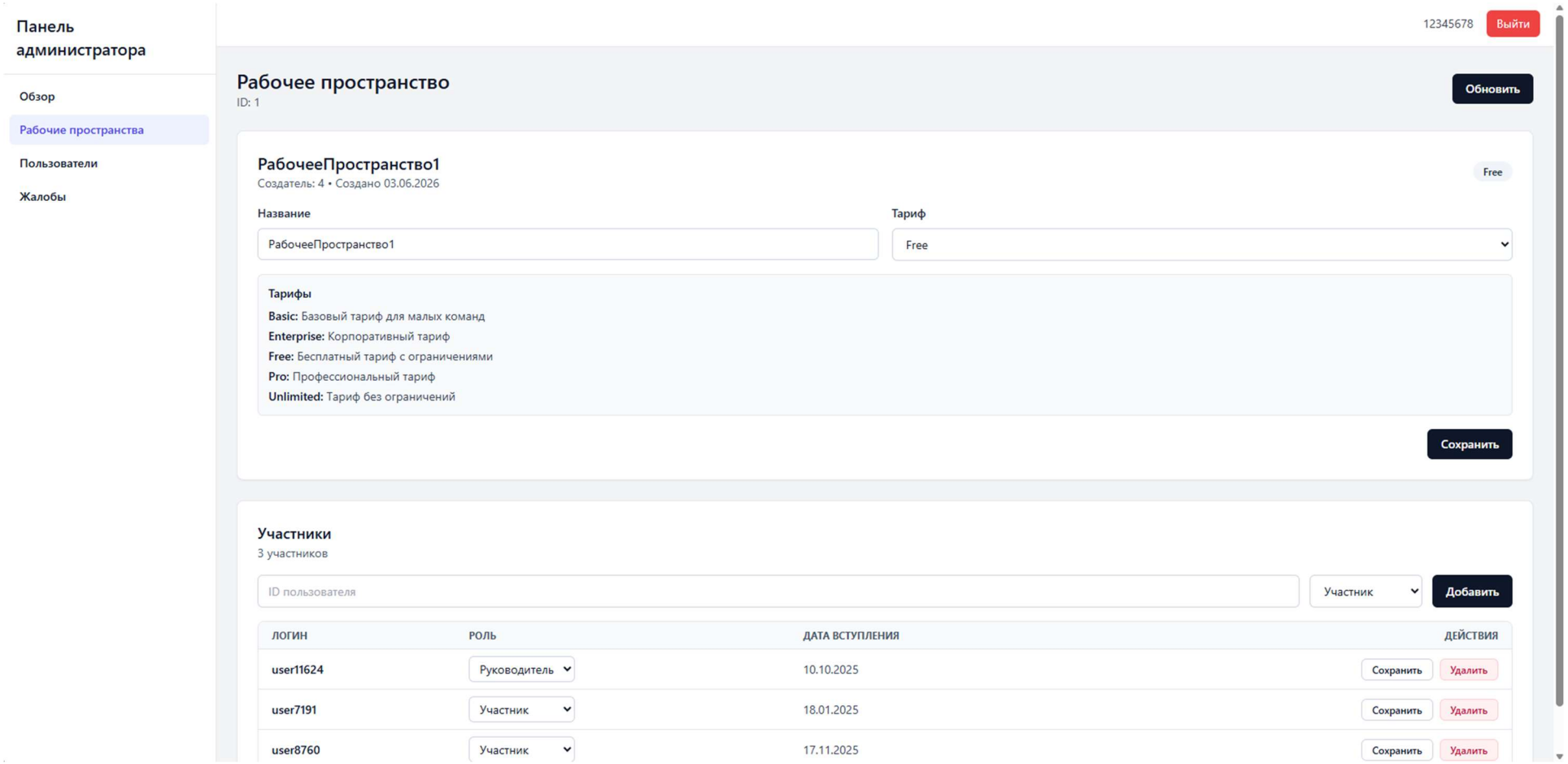

Рисунок Б.5 – Окно просмотра информации о рабочем пространстве

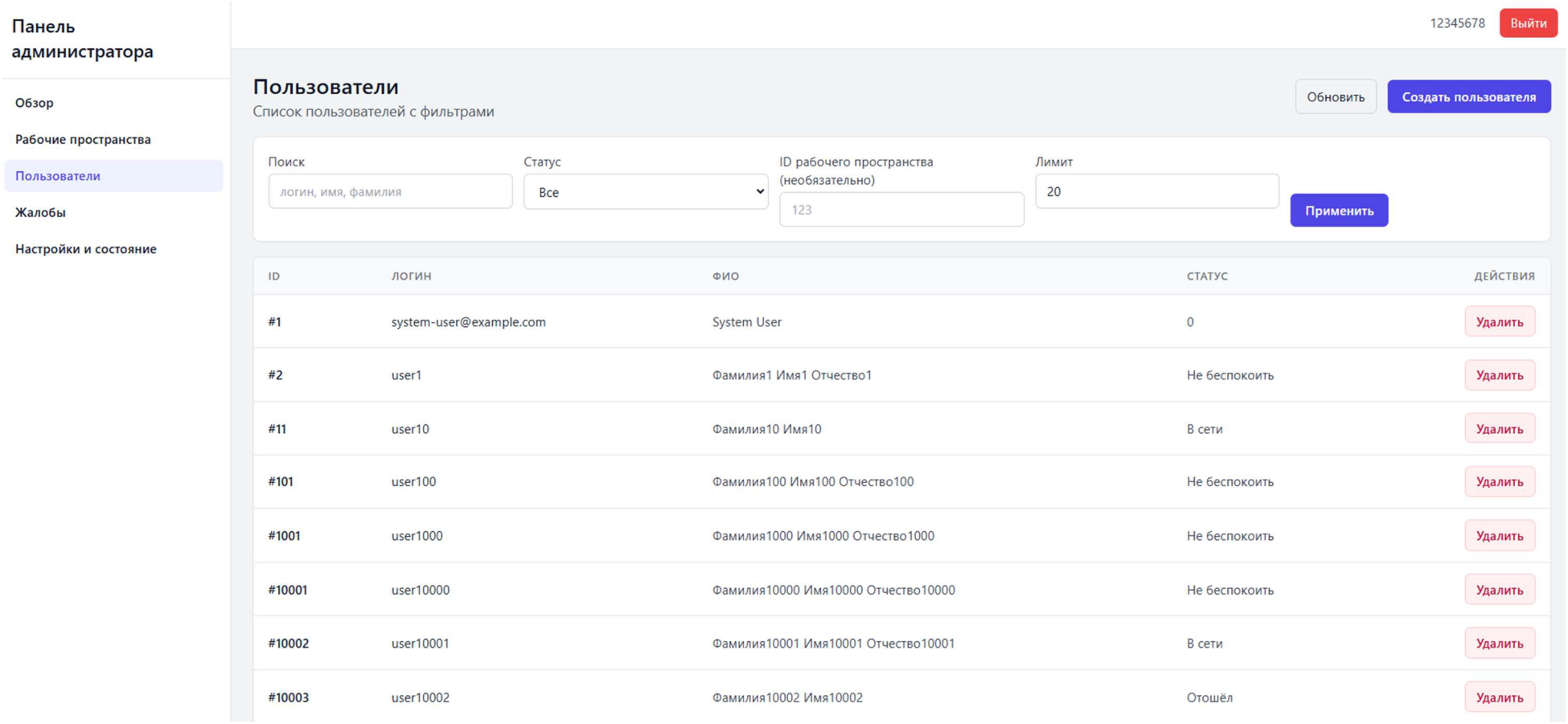

Рисунок Б.6 – Интерфейс раздела пользователей

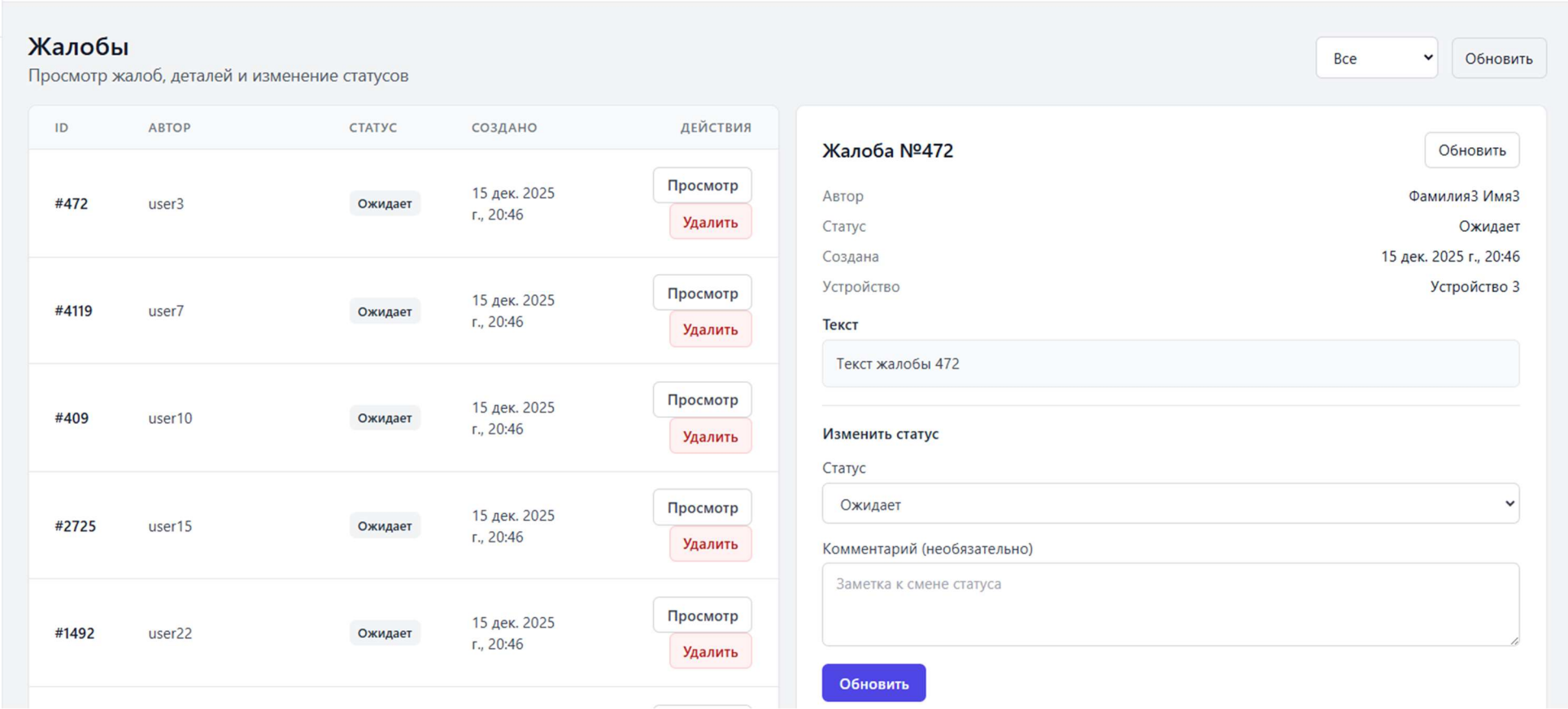

Рисунок Б.7 – Интерфейс раздела жалоб

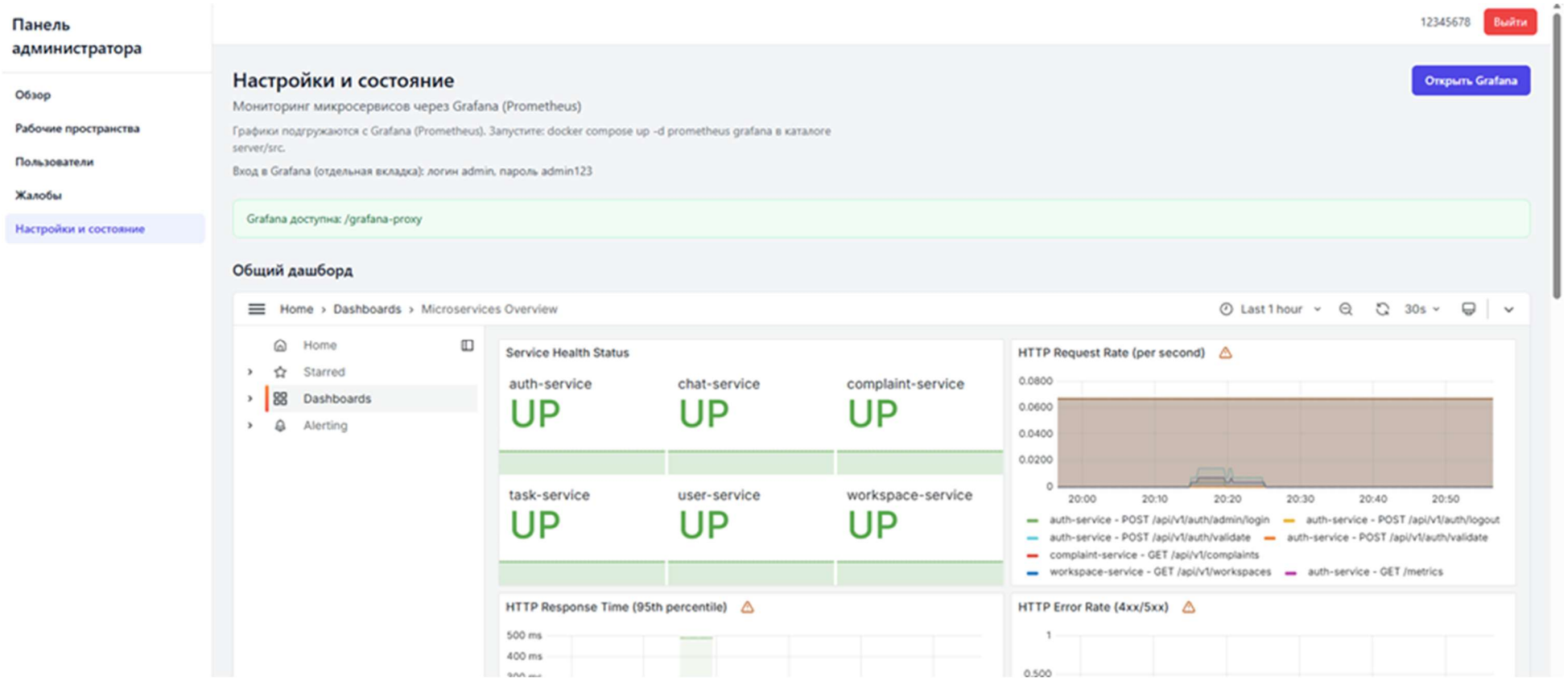

Рисунок Б.8 – Интерфейс страницы мониторинга

Приложение В

(обязательное)

Листинг клиентского приложения

Листинг В.1 – Файл ComplaintsPage.tsx

```
import { useEffect, useMemo, useState } from "react";
import { useMutation, useQuery, useQueryClient } from
"@tanstack/react-query";
import { apiFetch } from "../shared/api/client";
import { complaintStatusLabel, LOCALE, ru } from
"../shared/i18n/ru";

type Complaint = {
  id: number;
  author_login?: string;
  status: string;
  created_at: string;
};

type ComplaintsResponse = {
  complaints: Complaint[];
  total: number;
  limit: number;
  offset: number;
};

type StatusHistory = {
  status: string;
  comment?: string;
  changed_at?: string;
};

type ComplaintDetail = {
  id: number;
  text: string;
  status: string;
  author_name?: string;
  author_login?: string;
  device_description?: string;
  created_at: string;
  status_history?: StatusHistory[];
};

const STATUS_FILTER_OPTIONS = [
  { value: "all", label: ru.complaints.filterAll },
  { value: "pending", label: ru.complaints.statusPending },
  { value: "in_progress", label: ru.complaints.statusInProgress },
  { value: "resolved", label: ru.complaints.statusResolved },
  { value: "rejected", label: ru.complaints.statusRejected }
];

const STATUS_UPDATE_OPTIONS = STATUS_FILTER_OPTIONS.filter((opt)
=> opt.value !== "all");

function formatDate(value: string) {
  const date = new Date(value);
```

```
  return Number.isNaN(date.getTime())
    ? "—"
    : date.toLocaleString(LOCALE, {
        year: "numeric",
        month: "short",
        day: "numeric",
        hour: "2-digit",
        minute: "2-digit"
      });
}

function ComplaintsPage() {
  const queryClient = useQueryClient();
  const [statusFilter, setStatusFilter] = useState("all");
  const [selectedId, setSelectedId] = useState<number |
null>(null);
  const [updateStatus, setUpdateStatus] = useState("pending");
  const [updateComment, setUpdateComment] = useState("");
  const [pendingDeleteId, setPendingDeleteId] = useState<number |
null>(null);

  const {
    data: complaintsData,
    isLoading: isComplaintsLoading,
    isFetching: isComplaintsFetching,
    error: complaintsError,
    refetch: refetchComplaints
  } = useQuery<ComplaintsResponse>({
    queryKey: ["complaints", statusFilter],
    queryFn: () => {
      const params = new URLSearchParams({
        limit: "20",
        offset: "0"
      });
      if (statusFilter !== "all") {
        params.set("status", statusFilter);
      }
      return
apiFetch<ComplaintsResponse>(`/api/v1/complaints?${params.toString
()}`);
    },
    placeholderData: (previousData) => previousData,
    staleTime: 30_000
  });

  const {
    data: detailData,
    isLoading: isDetailLoading,
    isFetching: isDetailFetching,
    error: detailError,
    refetch: refetchDetail
  } = useQuery<ComplaintDetail>({
```

```
    queryKey: ["complaint", selectedId],
    queryFn: () =>
apiFetch<ComplaintDetail>(`/api/v1/complaints/${selectedId!}`),
    enabled: selectedId !== null,
    staleTime: 30_000
  });

  useEffect(() => {
    if (detailData?.status) {
      setUpdateStatus(detailData.status);
      setUpdateComment("");
    }
  }, [detailData?.status, detailData?.id]);

  const updateMutation = useMutation({
    mutationFn: (payload: { id: number; status: string; comment?:
string }) =>
      apiFetch<void>(`/api/v1/complaints/${payload.id}/status`, {
        method: "PUT",
        body: {
          status: payload.status,
          comment: payload.comment?.trim() || undefined
        }
      }),
    onSuccess: (_, variables) => {
      queryClient.invalidateQueries({ queryKey: ["complaints"] });
      queryClient.invalidateQueries({
        queryKey: ["complaint", variables.id]
      });
    }
  });

  useEffect(() => {
    updateMutation.reset();
  }, [selectedId, updateMutation]);

  const deleteMutation = useMutation({
    mutationFn: (id: number) =>
apiFetch<void>(`/api/v1/complaints/${id}`, { method: "DELETE" }),
    onMutate: (id) => {
      setPendingDeleteId(id);
    },
    onSettled: () => {
      setPendingDeleteId(null);
    },
    onSuccess: (_, id) => {
      queryClient.invalidateQueries({ queryKey: ["complaints"] });
      if (selectedId === id) {
        setSelectedId(null);
      } else {
        queryClient.invalidateQueries({ queryKey: ["complaint",
id] });
```

```
      }
    }
  });

  const handleUpdate = () => {
    if (selectedId === null) return;
    updateMutation.mutate({
      id: selectedId,
      status: updateStatus,
      comment: updateComment || undefined
    });
  };

  const handleDelete = (id: number) => {
    deleteMutation.mutate(id);
  };

  const authorLabel = useMemo(() => {
    if (!detailData) return "—";
    return detailData.author_name || detailData.author_login || "—
";
  }, [detailData]);

  return (
    <div className="p-6 space-y-4">
      <div className="flex flex-wrap items-center justify-between
gap-3">
        <div>
          <h1 className="text-2xl font-
semibold">{ru.complaints.title}</h1>
          <p className="text-gray-
600">{ru.complaints.subtitle}</p>
          {isComplaintsFetching && !isComplaintsLoading && (
            <p className="mt-1 text-sm text-indigo-600">⏳
{ru.actions.refreshing}</p>
          )}
        </div>
        <div className="flex items-center gap-3">
          <select
            value={statusFilter}
            onChange={(e) => setStatusFilter(e.target.value)}
            className="px-3 py-2 rounded-md border border-gray-300
text-sm bg-white"
          >
            {STATUS_FILTER_OPTIONS.map((option) => (
              <option key={option.value} value={option.value}>
                {option.label}
              </option>
            ))}
          </select>
          <button
            onClick={() => refetchComplaints()}
```

```
            disabled={isComplaintsFetching}
            className="px-3 py-2 rounded-md border border-gray-300
text-sm hover:bg-gray-50 disabled:opacity-50"
          >
            {isComplaintsFetching ? ru.actions.refreshing :
ru.actions.refresh}
          </button>
        </div>
      </div>

      {isComplaintsLoading && (
        <div className="bg-white border border-gray-200 rounded-lg
p-6 shadow-sm">
          <p className="text-gray-600">{ru.complaints.loading}</p>
        </div>
      )}

      {complaintsError && !isComplaintsLoading && (
        <div className="bg-red-50 border border-red-200 rounded-lg
p-4 text-red-700">
          <div className="flex items-center justify-between gap-
3">
            <p className="text-sm">
              {(complaintsError as Error).message ||
ru.errors.loadComplaints}
            </p>
            <button
              onClick={() => refetchComplaints()}
              className="px-3 py-1.5 rounded-md bg-red-600 text-
white text-sm hover:bg-red-700"
            >
              {ru.actions.retry}
            </button>
          </div>
        </div>
      )}

      {!isComplaintsLoading && !complaintsError && (
        <div className="grid gap-4 lg:grid-cols-2">
          <div className="bg-white border border-gray-200 rounded-
lg shadow-sm overflow-hidden">
            {complaintsData?.complaints?.length ? (
              <div className="overflow-x-auto">
                <table className="min-w-full divide-y divide-gray-
200">
                  <thead className="bg-gray-50">
                    <tr>
                      <th className="px-6 py-3 text-left text-xs
font-semibold text-gray-500 uppercase tracking-wider">
                        {ru.table.id}
                      </th>
```

```
                        <th className="px-6 py-3 text-left text-xs
font-semibold text-gray-500 uppercase tracking-wider">
                          {ru.table.author}
                        </th>
                        <th className="px-6 py-3 text-left text-xs
font-semibold text-gray-500 uppercase tracking-wider">
                          {ru.table.status}
                        </th>
                        <th className="px-6 py-3 text-left text-xs
font-semibold text-gray-500 uppercase tracking-wider">
                          {ru.table.created}
                        </th>
                        <th className="px-6 py-3 text-right text-xs
font-semibold text-gray-500 uppercase tracking-wider">
                          {ru.complaints.actions}
                        </th>
                      </tr>
                    </thead>
                    <tbody className="bg-white divide-y divide-gray-
200">
                      {complaintsData.complaints.map((complaint) =>
(
                        <tr key={complaint.id} className="hover:bg-
gray-50">
                          <td className="px-6 py-4 text-sm font-
medium text-gray-900">
                            #{complaint.id}
                          </td>
                          <td className="px-6 py-4 text-sm text-
gray-700">
                            {complaint.author_login || "—"}
                          </td>
                          <td className="px-6 py-4 text-sm">
                            <span className="inline-flex items-
center px-2 py-1 rounded-md text-xs font-medium bg-gray-100 text-
gray-800">

{complaintStatusLabel(complaint.status)}
                            </span>
                          </td>
                          <td className="px-6 py-4 text-sm text-
gray-700">
                            {formatDate(complaint.created_at)}
                          </td>
                          <td className="px-6 py-4 text-right space-
x-2">
                            <button
                              onClick={() =>
setSelectedId(complaint.id)}
                              className="px-3 py-1.5 rounded-md
border border-gray-300 text-sm font-medium text-gray-700 hover:bg-
gray-50"
```

```
                            >
                              {ru.actions.view}
                            </button>
                            <button
                              onClick={() =>
handleDelete(complaint.id)}
                              disabled={
                                deleteMutation.isPending &&
pendingDeleteId === complaint.id
                              }
                              className="px-3 py-1.5 rounded-md
border border-red-200 bg-red-50 text-sm font-medium text-red-700
hover:bg-red-100 disabled:opacity-50"
                            >
                              {deleteMutation.isPending &&
pendingDeleteId === complaint.id
                                ? ru.actions.deleting
                                : ru.actions.delete}
                            </button>
                          </td>
                        </tr>
                      ))}
                    </tbody>
                  </table>
                </div>
              ) : (
                <div className="p-6 text-center text-gray-
600">{ru.complaints.none}</div>
              )}
            </div>

            <div className="bg-white border border-gray-200 rounded-
lg shadow-sm p-6">
              {!selectedId && <p className="text-gray-
600">{ru.complaints.selectHint}</p>}

              {selectedId && (
                <div className="space-y-4">
                  <div className="flex items-center justify-
between">
                    <div>
                      <h2 className="text-lg font-
semibold">{ru.complaints.complaint(selectedId)}</h2>
                      {isDetailFetching && !isDetailLoading && (
                        <p className="text-sm text-indigo-
600">{ru.actions.refreshing}</p>
                      )}
                    </div>
                    <button
                      onClick={() => refetchDetail()}
                      disabled={isDetailFetching}
```

```
                    className="px-3 py-1.5 rounded-md border
border-gray-300 text-sm hover:bg-gray-50 disabled:opacity-50"
                  >
                    {isDetailFetching ? ru.actions.refreshing :
ru.actions.refresh}
                  </button>
                </div>

                {isDetailLoading && <p className="text-gray-
600">{ru.complaints.loadingDetails}</p>}

                {detailError && !isDetailLoading && (
                  <div className="bg-red-50 border border-red-200
rounded-md p-3 text-red-700">
                    <div className="flex items-center justify-
between gap-2">
                      <span className="text-sm">
                        {(detailError as Error).message ||
ru.errors.loadComplaint}
                      </span>
                      <button
                        onClick={() => refetchDetail()}
                        className="px-2.5 py-1 rounded-md bg-red-
600 text-white text-xs hover:bg-red-700"
                      >
                        {ru.actions.retry}
                      </button>
                    </div>
                  </div>
                )}

                {detailData && !detailError && (
                  <div className="space-y-4">
                    <div className="grid gap-2 text-sm">
                      <div className="flex justify-between">
                        <span className="text-gray-
500">{ru.complaints.author}</span>
                        <span className="text-gray-
900">{authorLabel}</span>
                      </div>
                      <div className="flex justify-between">
                        <span className="text-gray-
500">{ru.complaints.status}</span>
                        <span className="text-gray-900">

{complaintStatusLabel(detailData.status)}
                        </span>
                      </div>
                      <div className="flex justify-between">
                        <span className="text-gray-
500">{ru.complaints.created}</span>
```

```
                        <span className="text-gray-
900">{formatDate(detailData.created_at)}</span>
                      </div>
                      {detailData.device_description && (
                        <div className="flex justify-between">
                          <span className="text-gray-
500">{ru.complaints.device}</span>
                          <span className="text-gray-900 text-
right">
                            {detailData.device_description}
                          </span>
                        </div>
                      )}
                    </div>

                    <div>
                      <p className="text-sm font-semibold text-
gray-800 mb-1">{ru.complaints.text}</p>
                      <p className="text-sm text-gray-700
whitespace-pre-line bg-gray-50 border border-gray-200 rounded-md
p-3">
                        {detailData.text}
                      </p>
                    </div>

                    {detailData.status_history?.length ? (
                      <div>
                        <p className="text-sm font-semibold text-
gray-800 mb-2">
                          {ru.complaints.statusHistory}
                        </p>
                        <div className="space-y-2">
                          {detailData.status_history.map((entry,
idx) => (
                            <div
                              key={`${entry.status}-${idx}`}
                              className="border border-gray-200
rounded-md p-3 bg-gray-50 text-sm"
                            >
                              <div className="flex items-center
justify-between">
                                <span className="font-medium text-
gray-900">

{complaintStatusLabel(entry.status)}
                                </span>
                                <span className="text-gray-600">
                                  {entry.changed_at ?
formatDate(entry.changed_at) : "—"}
                                </span>
                              </div>
                              {entry.comment && (
```

```
                                <p className="mt-1 text-gray-700
whitespace-pre-line">
                                  {entry.comment}
                                </p>
                              )}
                            </div>
                          ))}
                        </div>
                      </div>
                    ) : null}

                    <div className="border-t border-gray-200 pt-4
space-y-3">
                      <p className="text-sm font-semibold text-
gray-800">{ru.complaints.updateStatus}</p>
                      <div className="grid gap-3">
                        <label className="text-sm text-gray-700">
                          {ru.complaints.status}
                          <select
                            value={updateStatus}
                            onChange={(e) =>
setUpdateStatus(e.target.value)}
                            className="mt-1 w-full px-3 py-2
rounded-md border border-gray-300 text-sm bg-white"
                          >
                            {STATUS_UPDATE_OPTIONS.map((option) =>
(
                              <option key={option.value}
value={option.value}>
                                {option.label}
                              </option>
                            ))}
                          </select>
                        </label>
                        <label className="text-sm text-gray-700">
                          {ru.complaints.commentOptional}
                          <textarea
                            value={updateComment}
                            onChange={(e) =>
setUpdateComment(e.target.value)}
                            rows={3}
                            className="mt-1 w-full px-3 py-2
rounded-md border border-gray-300 text-sm bg-white"

placeholder={ru.complaints.commentPlaceholder}
                          />
                        </label>
                        <button
                          onClick={handleUpdate}
                          disabled={updateMutation.isPending}
```

```
                            className="w-fit px-4 py-2 rounded-md
bg-indigo-600 text-white text-sm font-medium hover:bg-indigo-700
disabled:opacity-50"
                          >
                            {updateMutation.isPending ?
ru.actions.updating : ru.actions.update}
                          </button>
                          {updateMutation.error && (
                            <p className="text-sm text-red-600">
                              {(updateMutation.error as
Error).message || ru.errors.updateStatusFailed}
                            </p>
                          )}
                          {updateMutation.isSuccess && (
                            <p className="text-sm text-green-
700">{ru.complaints.statusUpdated}</p>
                          )}
                        </div>
                      </div>
                    </div>
                  )}
                </div>
              )}
            </div>
          </div>
        )}
      </div>
    );
}

export default ComplaintsPage;
```

Листинг В.2 – Файл WorkspaceDetailPage.tsx

```
import type { FormEvent } from "react";
import { useEffect, useMemo, useState } from "react";
import { useMutation, useQuery, useQueryClient } from
"@tanstack/react-query";
import { useParams } from "react-router-dom";
import { apiFetch } from "../shared/api/client";
import { LOCALE, ru } from "../shared/i18n/ru";
import { useAuthStore } from "../shared/state/auth";

type Tariff = {
  id: number;
  name: string;
  description?: string;
};

type TariffsResponse = {
  tariffs: Tariff[];
};
```

```
type WorkspaceDetails = {
  id: number;
  name: string;
  creator: number;
  created_at: string;
  tariff: Tariff | null;
};

type WorkspaceUpdateRequest = {
  name: string;
  tariff_id: number;
};

type Member = {
  user_id: number;
  login: string;
  role: number;
  joined_at: string;
};

type MembersResponse = {
  members: Member[];
};

const roleOptions = [
  { value: 2, label: "Руководитель" },
  { value: 1, label: "Участник" }
];

function WorkspaceDetailPage() {
  const { id: workspaceId } = useParams<{ id: string }>();
  const queryClient = useQueryClient();
  const { admin } = useAuthStore();

  const [workspaceForm, setWorkspaceForm] = useState({ name: "",
tariffId: "" });
  const [addMemberForm, setAddMemberForm] = useState({ userId: "",
role: "1" });
  const [roleDrafts, setRoleDrafts] = useState<Record<number,
string>>({});

  const workspaceQuery = useQuery({
    queryKey: ["workspace", workspaceId],
    queryFn: () =>
apiFetch<WorkspaceDetails>(`/api/v1/workspaces/${workspaceId}`),
    enabled: Boolean(workspaceId)
  });

  const tariffsQuery = useQuery({
    queryKey: ["workspaceTariffs"],
    queryFn: () =>
apiFetch<TariffsResponse>("/api/v1/workspaces/tariffs")
```

```
  });

  const membersQuery = useQuery({
    queryKey: ["workspaceMembers", workspaceId],
    queryFn: () =>
apiFetch<MembersResponse>(`/api/v1/workspaces/${workspaceId}/membe
rs`),
    enabled: Boolean(workspaceId)
  });

  const sortedTariffs = useMemo(
    () => tariffsQuery.data?.tariffs?.slice().sort((a, b) =>
a.name.localeCompare(b.name, "ru")) ?? [],
    [tariffsQuery.data]
  );

  useEffect(() => {
    if (workspaceQuery.data) {
      setWorkspaceForm({
        name: workspaceQuery.data.name,
        tariffId: workspaceQuery.data.tariff?.id ?
String(workspaceQuery.data.tariff.id) : ""
      });
    }
  }, [workspaceQuery.data]);

  const updateWorkspace = useMutation({
    mutationFn: (payload: WorkspaceUpdateRequest) =>
      apiFetch(`/api/v1/workspaces/${workspaceId}`, { method:
"PUT", body: payload }),
    onSuccess: () => {
      queryClient.invalidateQueries({ queryKey: ["workspace",
workspaceId] });
    }
  });

  const addMember = useMutation({
    mutationFn: (payload: { user_id: number; role: number;
leader_id: number }) =>
      apiFetch(`/api/v1/workspaces/${workspaceId}/members`, {
method: "POST", body: payload }),
    onSuccess: () => {
      setAddMemberForm({ userId: "", role: "1" });
      queryClient.invalidateQueries({ queryKey:
["workspaceMembers", workspaceId] });
    }
  });

  const updateMemberRole = useMutation({
    mutationFn: (payload: { userId: number; role: number;
leader_id: number }) =>
```

```
apiFetch(`/api/v1/workspaces/${workspaceId}/members/${payload.user
Id}`, {
        method: "PUT",
        body: { role: payload.role, leader_id: payload.leader_id }
      }),
    onSuccess: () => {
      queryClient.invalidateQueries({ queryKey:
["workspaceMembers", workspaceId] });
    }
  });

  const removeMember = useMutation({
    mutationFn: (userId: number) =>

apiFetch(`/api/v1/workspaces/${workspaceId}/members/${userId}`, {
method: "DELETE" }),
    onSuccess: () => {
      queryClient.invalidateQueries({ queryKey:
["workspaceMembers", workspaceId] });
    }
  });

  if (!workspaceId) {
    return (
      <div className="p-6">
        <div className="rounded-lg border border-amber-200 bg-
amber-50 px-4 py-3 text-amber-800">
          {ru.workspaceDetail.missingId}
        </div>
      </div>
    );
  }

  const formatDate = (value?: string) => {
    if (!value) return "—";
    const parsed = new Date(value);
    if (Number.isNaN(parsed.getTime())) return "—";
    return parsed.toLocaleDateString(LOCALE);
  };

  const handleUpdateWorkspace = (event: FormEvent) => {
    event.preventDefault();
    if (!workspaceForm.name.trim() || !workspaceForm.tariffId)
return;

    updateWorkspace.mutate({
      name: workspaceForm.name.trim(),
      tariff_id: Number(workspaceForm.tariffId)
    });
  };
```

```
  const handleAddMember = (event: FormEvent) => {
    event.preventDefault();
    if (!addMemberForm.userId.trim() || !admin?.id) return;

    addMember.mutate({
      user_id: Number(addMemberForm.userId),
      role: Number(addMemberForm.role),
      leader_id: Number(admin.id)
    });
  };

  return (
    <div className="p-6 space-y-6">
      <div className="flex items-center justify-between gap-3">
        <div>
          <h1 className="text-2xl font-semibold text-slate-
900">{ru.workspaceDetail.title}</h1>
          <p className="text-sm text-slate-600">ID:
{workspaceId}</p>
        </div>
        <div className="flex gap-2">
          <button
            type="button"
            onClick={() => {
              workspaceQuery.refetch();
              tariffsQuery.refetch();
              membersQuery.refetch();
            }}
            className="rounded-md bg-slate-900 px-4 py-2 text-sm
font-medium text-white hover:bg-slate-800 disabled:opacity-60"
            disabled={workspaceQuery.isFetching ||
tariffsQuery.isFetching || membersQuery.isFetching}
          >
            {ru.actions.refresh}
          </button>
        </div>
      </div>

      {workspaceQuery.isLoading ? (
        <div className="rounded-lg border border-slate-200 bg-
white px-4 py-3 text-slate-700">
          {ru.workspaceDetail.loading}
        </div>
      ) : workspaceQuery.error ? (
        <div className="space-y-3 rounded-lg border border-rose-
200 bg-rose-50 px-4 py-3 text-rose-700">
          <p>{(workspaceQuery.error as Error).message}</p>
          <button
            type="button"
            onClick={() => workspaceQuery.refetch()}
            className="rounded-md bg-rose-600 px-3 py-1.5 text-sm
text-white hover:bg-rose-700"
```

```
          >
            {ru.actions.retry}
          </button>
        </div>
      ) : workspaceQuery.data ? (
        <div className="space-y-6">
          <div className="rounded-lg border border-slate-200 bg-
white p-6 shadow-sm">
            <div className="mb-4 flex items-center justify-
between">
              <div>
                <h2 className="text-xl font-semibold text-slate-
900">{workspaceQuery.data.name}</h2>
                <p className="text-sm text-slate-600">
                  {ru.workspaceDetail.creatorCreated(
                    workspaceQuery.data.creator,
                    formatDate(workspaceQuery.data.created_at)
                  )}
                </p>
              </div>
              {workspaceQuery.data.tariff && (
                <span className="rounded-full bg-slate-100 px-3
py-1 text-xs font-medium text-slate-800">
                  {workspaceQuery.data.tariff.name}
                </span>
              )}
            </div>

            <form onSubmit={handleUpdateWorkspace}
className="space-y-4">
              <div className="grid gap-4 md:grid-cols-2">
                <div className="flex flex-col gap-2">
                  <label className="text-sm font-medium text-
slate-800">{ru.workspaceDetail.name}</label>
                  <input
                    value={workspaceForm.name}
                    onChange={(e) => setWorkspaceForm((prev) => ({
...prev, name: e.target.value }))}
                    className="w-full rounded-md border border-
slate-300 px-3 py-2 text-sm focus:border-slate-500 focus:outline-
none"

placeholder={ru.workspaceDetail.namePlaceholder}
                  />
                </div>
                <div className="flex flex-col gap-2">
                  <label className="text-sm font-medium text-
slate-800">{ru.workspaceDetail.tariff}</label>
                  {tariffsQuery.isLoading ? (
                    <p className="text-sm text-slate-
600">{ru.workspaceDetail.loadingTariffs}</p>
                  ) : tariffsQuery.error ? (
```

```
                  <div className="flex items-center gap-3">
                    <p className="text-sm text-rose-
700">{ru.workspaceDetail.loadTariffsFailed}</p>
                    <button
                      type="button"
                      onClick={() => tariffsQuery.refetch()}
                      className="rounded-md bg-rose-600 px-3 py-
1.5 text-sm text-white hover:bg-rose-700"
                    >
                      {ru.actions.retry}
                    </button>
                  </div>
                ) : (
                  <select
                    value={workspaceForm.tariffId}
                    onChange={(e) => setWorkspaceForm((prev) =>
({ ...prev, tariffId: e.target.value }))}
                    className="w-full rounded-md border border-
slate-300 px-3 py-2 text-sm focus:border-slate-500 focus:outline-
none"
                  >
                    <option
value="">{ru.workspaceDetail.selectTariff}</option>
                    {sortedTariffs.map((tariff) => (
                      <option key={tariff.id} value={tariff.id}>
                        {tariff.name}
                      </option>
                    ))}
                  </select>
                )}
              </div>
            </div>

            {sortedTariffs.length > 0 && (
              <div className="rounded-md border border-slate-200
bg-slate-50 p-3 text-sm text-slate-700">
                <div className="font-medium text-slate-
900">{ru.workspaceDetail.tariffs}</div>
                <ul className="mt-2 space-y-1">
                  {sortedTariffs.map((tariff) => (
                    <li key={tariff.id}>
                      <span className="font-semibold text-slate-
900">{tariff.name}:</span>{" "}
                      {tariff.description ||
ru.workspaceCreate.noDescription}
                    </li>
                  ))}
                </ul>
              </div>
            )}

            <div className="flex justify-end">
```

```
                <button
                  type="submit"
                  disabled={updateWorkspace.isPending}
                  className="rounded-md bg-slate-900 px-4 py-2
text-sm font-medium text-white hover:bg-slate-800
disabled:opacity-60"
                >
                  {updateWorkspace.isPending ? ru.actions.saving :
ru.actions.save}
                </button>
              </div>
            </form>
          </div>

          <div className="space-y-4 rounded-lg border border-
slate-200 bg-white p-6 shadow-sm">
            <div className="flex items-center justify-between">
              <div>
                <h3 className="text-lg font-semibold text-slate-
900">{ru.workspaceDetail.members}</h3>
                <p className="text-sm text-slate-600">

{ru.workspaceDetail.membersCount(membersQuery.data?.members?.lengt
h ?? 0)}
                </p>
              </div>
              {membersQuery.isFetching && (
                <span className="text-xs text-slate-
500">{ru.actions.refreshing}</span>
              )}
            </div>

            <form onSubmit={handleAddMember} className="grid gap-3
md:grid-cols-[1fr,auto,auto]">
              <input
                type="number"
                min={1}
                value={addMemberForm.userId}
                onChange={(e) => setAddMemberForm((prev) => ({
...prev, userId: e.target.value }))}
                className="rounded-md border border-slate-300 px-3
py-2 text-sm focus:border-slate-500 focus:outline-none"
                placeholder={ru.workspaceDetail.userIdPlaceholder}
              />
              <select
                value={addMemberForm.role}
                onChange={(e) => setAddMemberForm((prev) => ({
...prev, role: e.target.value }))}
                className="rounded-md border border-slate-300 px-3
py-2 text-sm focus:border-slate-500 focus:outline-none"
              >
                {roleOptions.map((role) => (
```

```
                  <option key={role.value} value={role.value}>
                    {role.label}
                  </option>
                ))}
              </select>
              <button
                type="submit"
                disabled={addMember.isPending}
                className="rounded-md bg-slate-900 px-4 py-2 text-
sm font-medium text-white hover:bg-slate-800 disabled:opacity-60"
              >
                {addMember.isPending ? ru.actions.adding :
ru.actions.add}
              </button>
            </form>

            {membersQuery.isLoading ? (
              <p className="text-sm text-slate-
600">{ru.workspaceDetail.loadingMembers}</p>
            ) : membersQuery.error ? (
              <div className="space-y-2 rounded-md border border-
rose-200 bg-rose-50 px-3 py-2 text-rose-700">
                <p>{(membersQuery.error as Error).message}</p>
                <button
                  type="button"
                  onClick={() => membersQuery.refetch()}
                  className="rounded-md bg-rose-600 px-3 py-1.5
text-xs text-white hover:bg-rose-700"
                >
                  {ru.actions.retry}
                </button>
              </div>
            ) : membersQuery.data &&
membersQuery.data.members.length > 0 ? (
              <div className="overflow-hidden rounded-lg border
border-slate-200">
                <table className="min-w-full divide-y divide-
slate-200 text-sm">
                  <thead className="bg-slate-50 text-xs uppercase
text-slate-600">
                    <tr>
                      <th className="px-4 py-2 text-
left">{ru.workspaceDetail.login}</th>
                      <th className="px-4 py-2 text-
left">{ru.workspaceDetail.role}</th>
                      <th className="px-4 py-2 text-
left">{ru.workspaceDetail.joined}</th>
                      <th className="px-4 py-2 text-
right">{ru.workspaceDetail.actions}</th>
                    </tr>
                  </thead>
```

```
                  <tbody className="divide-y divide-slate-200 bg-
white">
                    {membersQuery.data.members.map((member) => {
                      const draft = roleDrafts[member.user_id] ??
String(member.role);
                      const updating =
                        updateMemberRole.isPending &&
                        updateMemberRole.variables?.userId ===
member.user_id;
                      const removing =
                        removeMember.isPending &&
removeMember.variables === member.user_id;

                      return (
                        <tr key={member.user_id}
className="hover:bg-slate-50">
                          <td className="px-4 py-2 font-medium
text-slate-900">{member.login}</td>
                          <td className="px-4 py-2">
                            <select
                              value={draft}
                              onChange={(e) =>
                                setRoleDrafts((prev) => ({
...prev, [member.user_id]: e.target.value }))
                              }
                              className="rounded-md border border-
slate-300 px-2 py-1 text-sm focus:border-slate-500 focus:outline-
none"
                            >
                              {roleOptions.map((role) => (
                                <option key={role.value}
value={role.value}>
                                  {role.label}
                                </option>
                              ))}
                            </select>
                          </td>
                          <td className="px-4 py-2 text-slate-
700">{formatDate(member.joined_at)}</td>
                          <td className="px-4 py-2 text-right">
                            <div className="flex justify-end gap-
2">
                              <button
                                type="button"
                                onClick={() =>
                                  updateMemberRole.mutate({
                                    userId: member.user_id,
                                    role: Number(draft),
                                    leader_id: Number(admin?.id)
                                  })
                                }
                                disabled={updating}
```

```
                                    className="rounded-md border
border-slate-300 px-3 py-1 text-xs font-medium text-slate-900
hover:bg-slate-100 disabled:opacity-60"
                                  >
                                    {updating ? ru.actions.saving :
ru.actions.save}
                                  </button>
                                  <button
                                    type="button"
                                    onClick={() =>
removeMember.mutate(member.user_id)}
                                    disabled={removing}
                                    className="rounded-md border
border-rose-200 bg-rose-50 px-3 py-1 text-xs font-medium text-
rose-700 hover:bg-rose-100 disabled:opacity-60"
                                  >
                                    {removing ? ru.actions.removing :
ru.actions.delete}
                                  </button>
                                </div>
                              </td>
                            </tr>
                          );
                        })}
                      </tbody>
                    </table>
                  </div>
                ) : (
                  <p className="text-sm text-slate-
600">{ru.workspaceDetail.noMembers}</p>
                )}
              </div>
            </div>
          ) : null}
        </div>
      );
}

export default WorkspaceDetailPage;
```

Приложение Г

(рекомендуемое)

Листинг инструментов тестирования

Листинг Г.1 – Модульное тестирование

```
import { beforeEach, describe, expect, it, vi } from "vitest";

import { fetchUsers } from "../shared/api/users";
import { apiFetch } from "../shared/api/client";

vi.mock("../shared/api/client", () => ({
  apiFetch: vi.fn(),
}));

const mockedApiFetch = vi.mocked(apiFetch);

function getRequestedUrl(): string {
  const firstCall = mockedApiFetch.mock.calls[0];
  if (!firstCall || typeof firstCall[0] !== "string") {
    throw new Error("apiFetch was not called with URL");
  }
  return firstCall[0];
}

describe("fetchUsers", () => {
  beforeEach(() => {
    mockedApiFetch.mockReset();
  });

  it("подставляет limit=20 и offset=0 по умолчанию", async () => {
    mockedApiFetch.mockResolvedValueOnce({ users: [], total: 0,
limit: 20, offset: 0 });

    await fetchUsers({});

    const url = getRequestedUrl();
    expect(url).toContain("/api/v1/users?");
    expect(url).toContain("limit=20");
    expect(url).toContain("offset=0");
  });

  it("добавляет search в URL, если параметр передан", async () =>
{
    mockedApiFetch.mockResolvedValueOnce({ users: [], total: 0,
limit: 20, offset: 0 });

    await fetchUsers({ search: "ivanov" });

    const url = getRequestedUrl();
    expect(url).toContain("search=ivanov");
  });

  it("добавляет workspace_id в URL, если передано число", async ()
=> {
    mockedApiFetch.mockResolvedValueOnce({ users: [], total: 0,
limit: 20, offset: 0 });
```

```
    await fetchUsers({ workspace_id: 42 });

    const url = getRequestedUrl();
    expect(url).toContain("workspace_id=42");
  });

  it("не добавляет workspace_id при отсутствии или нечисловом
значении", async () => {
    mockedApiFetch.mockResolvedValue({ users: [], total: 0, limit:
20, offset: 0 });

    await fetchUsers({});
    expect(getRequestedUrl()).not.toContain("workspace_id=");

    mockedApiFetch.mockClear();
    await fetchUsers({ workspace_id: "abc" as unknown as number
});
    expect(getRequestedUrl()).not.toContain("workspace_id=");
  });

  it("добавляет status в URL, если передано число", async () => {
    mockedApiFetch.mockResolvedValueOnce({ users: [], total: 0,
limit: 20, offset: 0 });

    await fetchUsers({ status: 3 });

    const url = getRequestedUrl();
    expect(url).toContain("status=3");
  });

  it("не включает status в URL при отсутствии или нечисловом
значении", async () => {
    mockedApiFetch.mockResolvedValue({ users: [], total: 0, limit:
20, offset: 0 });

    await fetchUsers({});
    expect(getRequestedUrl()).not.toContain("status=");

    mockedApiFetch.mockClear();
    await fetchUsers({ status: "bad" as unknown as number });
    expect(getRequestedUrl()).not.toContain("status=");
  });

  it("корректно передает все параметры одновременно", async () =>
{
    mockedApiFetch.mockResolvedValueOnce({ users: [], total: 1,
limit: 50, offset: 10 });

    await fetchUsers({
      search: "petrov",
      workspace_id: 7,
```

```
      status: 2,
      limit: 50,
      offset: 10,
    });

    const url = getRequestedUrl();
    expect(url).toContain("search=petrov");
    expect(url).toContain("workspace_id=7");
    expect(url).toContain("status=2");
    expect(url).toContain("limit=50");
    expect(url).toContain("offset=10");
  });

  it("возвращает объект, полученный от apiFetch", async () => {
    const result = { users: [{ id: 1, login: "admin" }], total: 1,
limit: 20, offset: 0 };
    mockedApiFetch.mockResolvedValueOnce(result);

    const response = await fetchUsers({ search: "admin" });

    expect(response).toBe(result);
  });

  it("пробрасывает ошибку от apiFetch без перехвата", async () =>
{
    const error = new Error("network failed");
    mockedApiFetch.mockRejectedValueOnce(error);

    await expect(fetchUsers({ search: "admin"
})).rejects.toThrow("network failed");
  });
});
```

Листинг Г.2 – Автоматизированный подсчет метрик

```
import fs from "node:fs";
import path from "node:path";
import ts from "typescript";
import { fileURLToPath } from "node:url";

const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);
const projectRoot = path.resolve(__dirname, "..");
const srcDir = path.join(projectRoot, "src");
const reportsDir = path.join(projectRoot, "reports");

const sourceExtensions = new Set([".ts", ".tsx"]);

/**
 * @typedef {Object} FunctionMetric
 * @property {number} index
 * @property {string} name
 * @property {string} kind
```

```
 * @property {string} location
 * @property {number} paramsCount
 * @property {boolean} hasReturnType
 * @property {number} bodyLines
 * @property {number} cyclomaticComplexity
 * @property {number} statementsCount
 * @property {number} maxNestingDepth
 */

/**
 * @param {string} dir
 * @returns {string[]}
 */
function collectSourceFiles(dir) {
  /** @type {string[]} */
  const result = [];

  for (const entry of fs.readdirSync(dir, { withFileTypes: true
})) {
    const fullPath = path.join(dir, entry.name);
    if (entry.isDirectory()) {
      result.push(...collectSourceFiles(fullPath));
      continue;
    }

    const ext = path.extname(entry.name);
    if (!sourceExtensions.has(ext) ||
entry.name.endsWith(".d.ts")) {
      continue;
    }
    result.push(fullPath);
  }

  return result;
}

/**
 * @param {ts.Node} node
 * @returns {boolean}
 */
function isFunctionLikeNode(node) {
  return (
    ts.isFunctionDeclaration(node) ||
    ts.isArrowFunction(node) ||
    ts.isFunctionExpression(node) ||
    ts.isMethodDeclaration(node) ||
    ts.isConstructorDeclaration(node) ||
    ts.isGetAccessorDeclaration(node) ||
    ts.isSetAccessorDeclaration(node)
  );
}
```

```
/**
 * @param {ts.FunctionLikeDeclarationBase} node
 * @returns {string}
 */
function getFunctionKind(node) {
  if (ts.isFunctionDeclaration(node)) return "function
declaration";
  if (ts.isArrowFunction(node)) return "arrow function";
  if (ts.isFunctionExpression(node)) return "function expression";
  if (ts.isMethodDeclaration(node)) return "method declaration";
  if (ts.isConstructorDeclaration(node)) return "constructor";
  if (ts.isGetAccessorDeclaration(node)) return "get accessor";
  if (ts.isSetAccessorDeclaration(node)) return "set accessor";
  return "unknown";
}

/**
 * @param {ts.FunctionLikeDeclarationBase} node
 * @returns {string}
 */
function getFunctionName(node) {
  if ("name" in node && node.name) {
    if (ts.isIdentifier(node.name)) return node.name.text;
    return node.name.getText();
  }

  const parent = node.parent;
  if (parent && ts.isVariableDeclaration(parent) &&
ts.isIdentifier(parent.name)) {
    return parent.name.text;
  }
  if (parent && ts.isPropertyAssignment(parent)) {
    return parent.name.getText();
  }
  if (parent && ts.isBinaryExpression(parent)) {
    return parent.left.getText();
  }
  return "<anonymous>";
}

/**
 * @param {ts.FunctionLikeDeclarationBase} fn
 * @param {ts.SourceFile} sourceFile
 * @returns {number}
 */
function getBodyLines(fn, sourceFile) {
  if (!fn.body) return 0;
  const bodyStart =
sourceFile.getLineAndCharacterOfPosition(fn.body.getStart(sourceFi
le)).line + 1;
```

```
  const bodyEnd =
sourceFile.getLineAndCharacterOfPosition(fn.body.getEnd()).line +
1;
  return Math.max(1, bodyEnd - bodyStart + 1);
}

/**
 * @param {ts.Node} node
 * @returns {boolean}
 */
function isNestingNode(node) {
  return (
    ts.isIfStatement(node) ||
    ts.isForStatement(node) ||
    ts.isForInStatement(node) ||
    ts.isForOfStatement(node) ||
    ts.isWhileStatement(node) ||
    ts.isDoStatement(node) ||
    ts.isSwitchStatement(node) ||
    ts.isCatchClause(node) ||
    ts.isConditionalExpression(node)
  );
}

/**
 * @param {ts.FunctionLikeDeclarationBase} fn
 * @returns {{ cyclomaticComplexity: number, statementsCount:
number, maxNestingDepth: number }}
 */
function analyzeBody(fn) {
  if (!fn.body) {
    return { cyclomaticComplexity: 1, statementsCount: 0,
maxNestingDepth: 0 };
  }

  let cyclomaticComplexity = 1;
  let statementsCount = 0;
  let maxNestingDepth = 0;

  /**
   * @param {ts.Node} node
   * @param {number} depth
   */
  function walk(node, depth) {
    if (node !== fn && isFunctionLikeNode(node)) {
      return;
    }

    if (ts.isStatement(node) && !ts.isBlock(node)) {
      statementsCount += 1;
    }
```

```
    if (
      ts.isIfStatement(node) ||
      ts.isForStatement(node) ||
      ts.isForInStatement(node) ||
      ts.isForOfStatement(node) ||
      ts.isWhileStatement(node) ||
      ts.isDoStatement(node) ||
      ts.isCatchClause(node) ||
      ts.isConditionalExpression(node)
    ) {
      cyclomaticComplexity += 1;
    }

    if (ts.isCaseClause(node)) {
      cyclomaticComplexity += 1;
    }

    if (
      ts.isBinaryExpression(node) &&
      (node.operatorToken.kind ===
ts.SyntaxKind.AmpersandAmpersandToken ||
        node.operatorToken.kind === ts.SyntaxKind.BarBarToken ||
        node.operatorToken.kind ===
ts.SyntaxKind.QuestionQuestionToken)
    ) {
      cyclomaticComplexity += 1;
    }

    const nextDepth = isNestingNode(node) ? depth + 1 : depth;
    if (nextDepth > maxNestingDepth) {
      maxNestingDepth = nextDepth;
    }

    node.forEachChild((child) => walk(child, nextDepth));
  }

  walk(fn.body, 0);

  return { cyclomaticComplexity, statementsCount, maxNestingDepth
};
}

/**
 * @param {FunctionMetric[]} metrics
 */
function writeReports(metrics) {
  fs.mkdirSync(reportsDir, { recursive: true });

  const jsonPath = path.join(reportsDir, "function-metrics.json");
  fs.writeFileSync(jsonPath, JSON.stringify(metrics, null, 2),
"utf-8");
```

```
  const csvHeader = [
    "index",
    "name",
    "kind",
    "location",
    "paramsCount",
    "hasReturnType",
    "bodyLines",
    "cyclomaticComplexity",
    "statementsCount",
    "maxNestingDepth",
  ];

  const csvRows = metrics.map((m) =>
    [
      m.index,
      JSON.stringify(m.name),
      JSON.stringify(m.kind),
      JSON.stringify(m.location),
      m.paramsCount,
      m.hasReturnType,
      m.bodyLines,
      m.cyclomaticComplexity,
      m.statementsCount,
      m.maxNestingDepth,
    ].join(",")
  );

  const csvPath = path.join(reportsDir, "function-metrics.csv");
  fs.writeFileSync(csvPath, [csvHeader.join(","),
...csvRows].join("\n"), "utf-8");
}

function main() {
  const files = collectSourceFiles(srcDir);
  /** @type {FunctionMetric[]} */
  const metrics = [];

  for (const filePath of files) {
    const sourceText = fs.readFileSync(filePath, "utf-8");
    const sourceFile = ts.createSourceFile(filePath, sourceText,
ts.ScriptTarget.Latest, true);

    /**
     * @param {ts.Node} node
     */
    function visit(node) {
      if (isFunctionLikeNode(node)) {
        const fn = /** @type {ts.FunctionLikeDeclarationBase} */
(node);
        const kind = getFunctionKind(fn);
        const name = getFunctionName(fn);
```

```
        const pos =
sourceFile.getLineAndCharacterOfPosition(fn.getStart(sourceFile));
        const relativePath = path.relative(srcDir,
filePath).replaceAll("\\", "/");
        const location = `${relativePath}:${pos.line +
1}:${pos.character + 1}`;
        const bodyLines = getBodyLines(fn, sourceFile);
        const { cyclomaticComplexity, statementsCount,
maxNestingDepth } = analyzeBody(fn);

        metrics.push({
          index: metrics.length,
          name,
          kind,
          location,
          paramsCount: fn.parameters.length,
          hasReturnType: Boolean(fn.type),
          bodyLines,
          cyclomaticComplexity,
          statementsCount,
          maxNestingDepth,
        });
      }
      node.forEachChild(visit);
    }

    visit(sourceFile);
  }

  writeReports(metrics);
}

main();
```

Приложение Д

(рекомендуемое)

Акт о внедрении

АКТ ВНЕДРЕНИЯ
программного продукта в эксплуатацию

г. Пенза «05» июня 2026 г.

Настоящий акт составлен о том, что программный продукт «Корпоративный мессенджер "КМ-1"», разработанный Сахно Никитой Сергеевичем и Новосельцев Александром Андреевичем в рамках выполнения дипломного проекта, был внедрён в эксплуатацию в ООО «Смартвей» по адресу: г. Пенза, ул. Московская, д. 39.

Программный продукт внедрён в деятельность Отдела разработки клиентских продуктов, а именно в работу Команды мобильной разработки. В ходе внедрения разработанное веб-приложение корпоративного мессенджера было успешно интегрировано в рабочие процессы команды и используется для организации внутренних коммуникаций, управления задачами и рабочими пространствами, а также повышения эффективности взаимодействия между участниками проектных групп.

По результатам эксплуатации установлено, что программный продукт функционирует корректно, соответствует поставленным задачам и может быть рекомендован для дальнейшего использования в производственной деятельности подразделения.

Ответственным за внедрение программного продукта является: Сахно Никита Сергеевич

Рисунок Д.1 – Первая страница акта о внедрении

Руководитель подразделения:
Моисеев Максим Геннадьевич.

Разработчик	Сахно Н.С.
Разработчик	Новосельцев А.А.
Руководитель подразделения	Моисеев М.Г.
Подпись / дата	

Рисунок Д.2 – Вторая страница акта о внедрении